MAGAZINE

BSD

FOR NOVICE AND ADVANCED USERS

Unix Security

FIND THE BEST SECURITY TOOL FOR YOU

BASIC UNIX QUEUING TECHNIQUES

HOW SECURE CAN SECURE SHELL (SSH) BE?

SNIFFING AND RECOVERING NETWORK INFORMATION

DYNAMIC MEMORY ALLOCATION

VOL.8 NO.01
ISSUE 01/2014(54)
1898-9144

Dear BSD Readers,

This January issue of BSD is devoted to Unix Security. *It is the beginning of the year so we think that all of us have made some New Year resolutions. I think that all of us want to be happy and feel secure and that is why we created this issue devoted to Unix Security.*

Inside this BSD issue, we collected the articles written by experts in that field to provide you with best-quality knowledge. Enjoy your reading and develop with our Magazine!

Inside this BSD issue, we publish the 3 articles by Mark Sitkowski. If you want to find out more on Unix security, you should read them all. We would like to highlight this one on Dynamic Memory Allocation in Unix Systems.

Also, we recommend that you read Phillip's article that will teach you how to use the Mac OS X hackers toolbox. This article can be extremely useful for all Mac users who aspire to be good security experts.

Of course, please do not forget to read the 3rd part of Arkadiusz's article on Virtual Private Networks supported by OpenSSH. And for dessert, please go to see what Rob wrote for you this time. We really like his column and are waiting for the next month eagerly.

However, as long as we have our precious readers, we have a purpose. We owe you a huge THANK YOU. Everything we do, we do with you on our minds. We are grateful for every comment and opinion, either positive or negative. Every word from you lets us improve BSD magazine and brings us closer to the ideal shape of our publication, or, we should say – your publication.

Thank you BSD fans for your invaluable support and contribution.

Ewa & BSD team

MAGAZINE BSD

Editor in Chief:
Ewa Dudzic
ewa.dudzic@software.com.pl

Contributing:
Michael Shirk, Andrey Vedikhin, Petr Topiarz, Charles Rapenne, Anton Borisov, Jeroen van Nieuwenhuizen, José B. Alós, Luke Marsden, Salih Khan, Arkadiusz Majewski, BEng

Top Betatesters & Proofreaders:
Annie Zhang, Denise Ebery, Eric Geissinger, Luca Ferrari, Imad Soltani, Olaoluwa Omokanwaye, Radjis Mahangoe, Mani Kanth, Ben Milman, Mark VonFange

Special Thanks:
Annie Zhang
Denise Ebery

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski
ireneusz.pogroszewski@software.com.pl

Senior Consultant/Publisher:
Paweł Marciniak
pawel@software.com.pl

CEO:
Ewa Dudzic
ewa.dudzic@software.com.pl

Production Director:
Andrzej Kuca
andrzej.kuca@software.com.pl

Publisher:
Hakin9 Media SK
02-676 Warsaw, Poland
Postepu 17D
Poland
worldwide publishing
editors@bsdmag.org
www.bsdmag.org

Hakin9 Media SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: editors@bsdmag.org.

Nmap: How to Use it

Nmap stands for "Network Mapper". It's been seen in many films like the Matrix Reloaded, Bourne Ultimatum, Die Hard 4, etc. When Nmap was created, it could only be used on the Linux Platform but now it supports all the major OSes like Linux, UNIX, Windows, and Mac OS platforms.

From the beginning its only job was to be a port scanner, but now it can do the following things: remote OS detection, Time based Scanning, Firewall Evasion Technique, The Scripting Engine, Multi-probe Ping Scanning, etc...

Installation of Nmap

For the installation of Nmap, go to *http://nmap.org/download.html*. On this page you can find the following options:

- Downloading Nmap
- Source Code Distribution (in case you wish to compile Nmap yourself)
- Microsoft Windows Binaries
- Linux RPM Source and Binaries
- Mac OS X Binaries
- Other Operating System

Installation on Windows

Select options as per your operating system. First, we'll see how to install it in Windows. Go to the Microsoft Windows Binaries. Now you can use Nmap in graphical mode as well as command-line. For the command Line download click on

```
Latest command-Line zipfile nmap-6.01-win32.zip +
```

For the Graphical Version click on:

```
Latest command-Line self-installer nmap-6.01-setup.exe +
```

When the download is completed, you can find the folder named nmap-6.0. First unzip the folder. After unzipping, you can find the 3 directories and 26 files. In the three directories named License, nselib and scripts, there are now four executable files: nmap, `winpcapnmap-4.12`, `vcedist2008_x86`, `vcredist_x86`. The fifth important file is `nmap_performance.reg` file and the others are supporting files for running nmap (there is also ncat, ndiff, nmapupdate, nping but now we are not going to discuss them).

After that first of all run the winpcap-nmap-4.12 and install the winpcap. Winpcap is a packet capture library. Then install vcedist2008_x86, vcredist_x86 and at last,

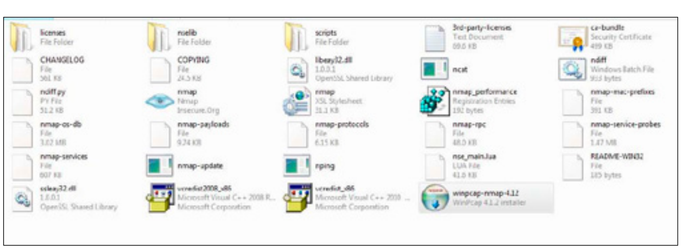


Figure 1. *All Unzip Files*

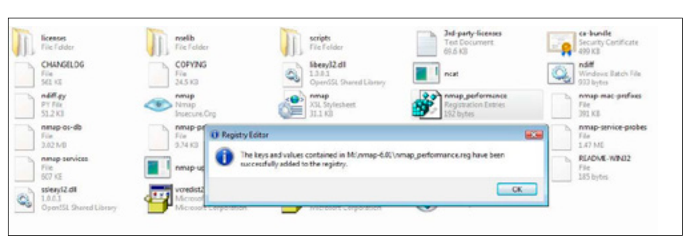


Figure 2. *Registy Entry*

double-click on the nmap_performance.reg file. This file is used for the entry in `[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters]`

```
"MaxUserPort"=dword:0000fffe +
```

In the Parameters section there will be an entry of `dword:0000fffe`, which is a hexadecimal value. In the binary it's 65534, which means the maximum user port is 65534:

```
"TcpTimedWaitDelay"=dword:0000001e + Tcp Timed
wait delay is 30, + "StrictTimeWaitSeqCheck"=dword:
00000001 and nmap is wait for the seq check is 1. +
```

Now you can use nmap in Windows. Go into the installed directory and give the simple command nmap 10.0.0.5. In the figure below, you can see the result.

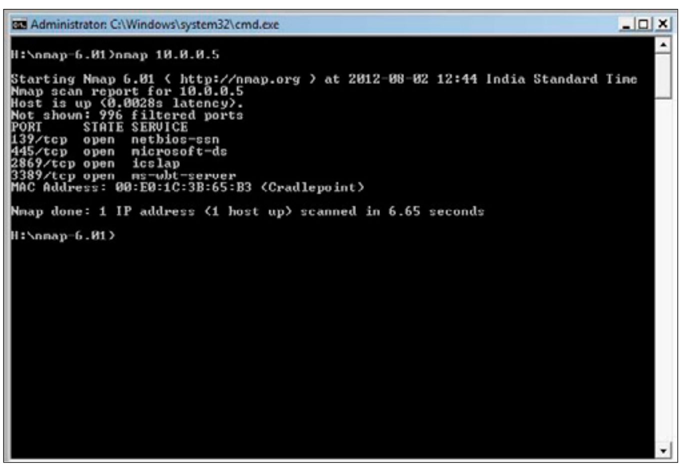


Figure 3. *Ready to use in windows*

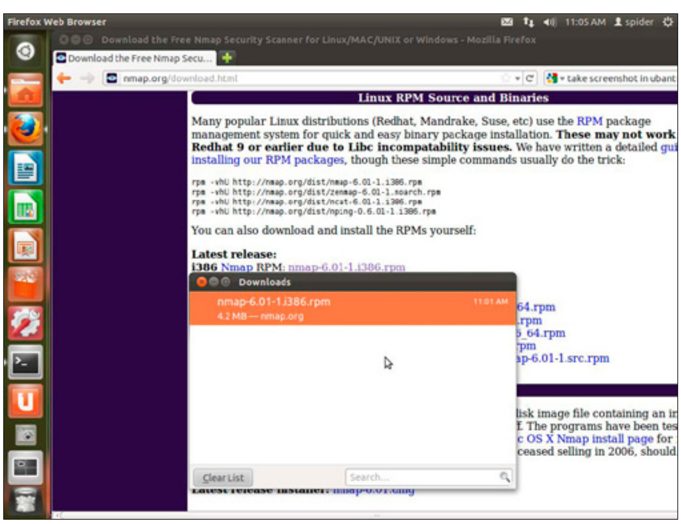


Figure 4. *Downloading the rpm file*

Installation on Ubuntu

Step 1: downloading from GUI

Visit the *http://nmap.org/download.html* for Linux – you can download it from the shell and from the GUI interface. Go to the 4th option, Linux RPM Source and Binaries, shown in Figure 4. Click on nmap-6.01-1.i386.rpm. Now it will download and you can see the 4.2 MB size.

Step 2: downloading from GUI

Go to the terminal and give the following command as shown in Figure 5.

```
wget http://nmap.org/dist/nmap-6.00.tar.bz2 +
```

After the download finishes, you can see the file named: `nmap-6.00.tar.bz2`.

Now you have to unzip this file by giving the command in Figure 6.

Command is `bzip2 -cd nmap-6.00.tar.bz2 | tar xvf` And then you have to run these commands ./configure, make, make install as a root.

Basic Scanning Technique

In the basic technique, we use Nmap without any switch. In this section we can see the flexibility of Nmap because it supports *classless Inter-Domain Routing* (CIDR) notation, octet ranges, DNS names, IPv6 addresses. So how can we scan multiple IPs?

Nmap gives the result in three titles. The first is PORT; it displays the port number or protocol. The second is STATE. There are six states that Nmap can result in:

- Open – Open State that means the application listening is active for TCP & UDP connection.
- Close – Close State means the application is not listening but they are accessible.
- Filtered – Filtered Filtered State means the port Responding is blocked by a packet filter; because of that it's hard to identify if the port is Open or not.
- Unfiltered – it's hard to determine for Nmap port if it is open or closed but they are accessible.

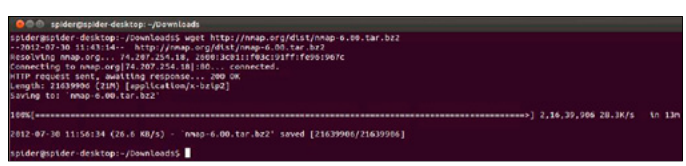


Figure 5. *Downloading from the shell mode*

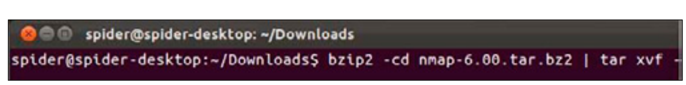


Figure 6. *Unzip nmap*

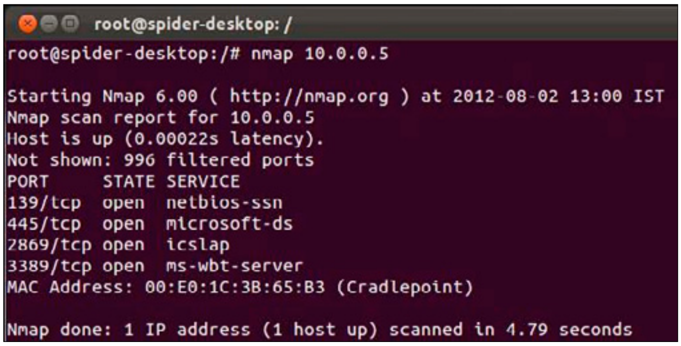
- Open – Filtered – this is the mutual state where you don't know if the port is open or not. You have to scan with techniques like Null, Fin, Xmas.
- Close – Filtered – Even in this state Nmap is not able to identify if the port is open or Closed. For information you have to scan the IP. ID idle scan only is the way to know more

This is the status of the port – Open or Closed. The third is SERVICE – which type of service is running on the port. In the last Nmap is shown a MAC address of the scanned system; how many hosts are up; how many times Nmap is consumed during scanning--most of this result shows in seconds.

Scanning a Single IP/Host/Domain

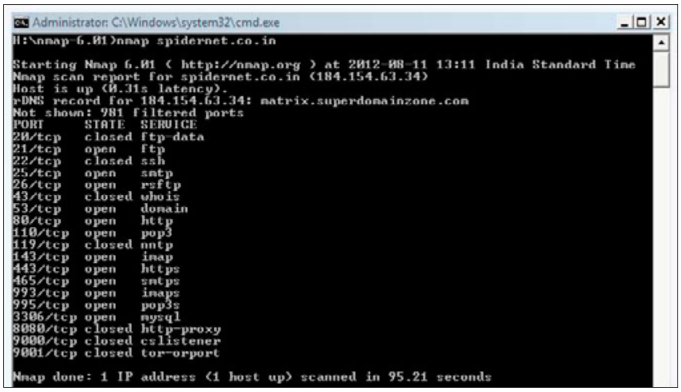
See Figure 7 & 8. Example:

```
#nmap <Live Domain/hostname/IP/Range of IP/Subnet>
#nmap 10.0.0.1
#nmap 10.0.0.1,2,3,4,5
#nmap 10.0.0.1-5
#nmap 10.0.0.0/8
#nmap 10.0.0.1 10.0.0.2 10.0.0.3 10.0.0.4 10.0.0.5
#nmap spider
#nmap spidernet.co.in
```



```
root@spider-desktop: /
root@spider-desktop:/# nmap 10.0.0.5

Starting Nmap 6.00 ( http://nmap.org ) at 2012-08-02 13:00 IST
Nmap scan report for 10.0.0.5
Host is up (0.00022s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
2869/tcp open  icslap
3389/tcp open  ms-wbt-server
MAC Address: 00:E0:1C:3B:65:B3 (Cradlepoint)

Nmap done: 1 IP address (1 host up) scanned in 4.79 seconds
```

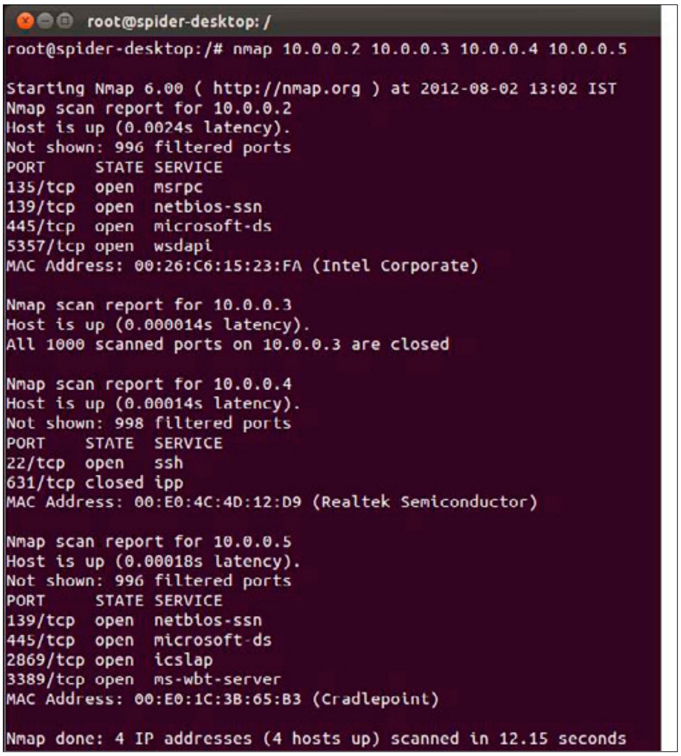
Figure 7. *Scanning single IP*



```
Administrator: C:\Windows\system32\cmd.exe
H:\nmap-6.01>nmap spidernet.co.in

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-11 13:11 India Standard Time
Nmap scan report for spidernet.co.in (184.154.63.34)
Host is up (0.31s latency).
rDNS record for 184.154.63.34: matrix.superdomainzone.com
Not shown: 981 filtered ports
PORT     STATE  SERVICE
20/tcp   closed ftp-data
21/tcp   open   ftp
22/tcp   closed ssh
25/tcp   open   smtp
26/tcp   open   rsftp
43/tcp   closed whois
53/tcp   open   domain
80/tcp   open   http
110/tcp  open   pop3
119/tcp  closed nntp
143/tcp  open   imap
443/tcp  open   https
465/tcp  open   smtps
993/tcp  open   imaps
995/tcp  open   pop3s
3306/tcp open   mysql
8080/tcp closed http-proxy
9000/tcp closed cslistener
9001/tcp closed tor-orport

Nmap done: 1 IP address (1 host up) scanned in 95.21 seconds
```

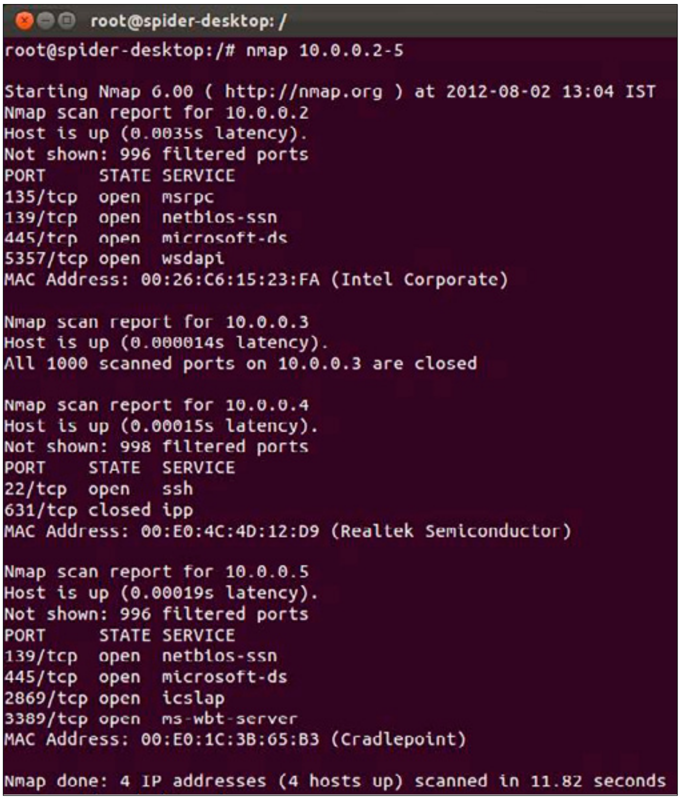
Figure 8. *Scanning domain*

By default, Nmap scans the 1000 most commonly used TCP/IP ports. If you can compare the result then you can



```
root@spider-desktop: /
root@spider-desktop:/# nmap 10.0.0.2 10.0.0.3 10.0.0.4 10.0.0.5

Starting Nmap 6.00 ( http://nmap.org ) at 2012-08-02 13:02 IST
Nmap scan report for 10.0.0.2
Host is up (0.0024s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
5357/tcp open  wsdapi
MAC Address: 00:26:C6:15:23:FA (Intel Corporate)

Nmap scan report for 10.0.0.3
Host is up (0.000014s latency).
All 1000 scanned ports on 10.0.0.3 are closed

Nmap scan report for 10.0.0.4
Host is up (0.00014s latency).
Not shown: 998 filtered ports
PORT    STATE  SERVICE
22/tcp  open   ssh
631/tcp closed ipp
MAC Address: 00:E0:4C:4D:12:D9 (Realtek Semiconductor)

Nmap scan report for 10.0.0.5
Host is up (0.00018s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
2869/tcp open  icslap
3389/tcp open  ms-wbt-server
MAC Address: 00:E0:1C:3B:65:B3 (Cradlepoint)

Nmap done: 4 IP addresses (4 hosts up) scanned in 12.15 seconds
```

Figure 9. *Multiple IP addresses*



```
root@spider-desktop: /
root@spider-desktop:/# nmap 10.0.0.2-5

Starting Nmap 6.00 ( http://nmap.org ) at 2012-08-02 13:04 IST
Nmap scan report for 10.0.0.2
Host is up (0.0035s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
5357/tcp open  wsdapi
MAC Address: 00:26:C6:15:23:FA (Intel Corporate)

Nmap scan report for 10.0.0.3
Host is up (0.000014s latency).
All 1000 scanned ports on 10.0.0.3 are closed

Nmap scan report for 10.0.0.4
Host is up (0.00015s latency).
Not shown: 998 filtered ports
PORT    STATE  SERVICE
22/tcp  open   ssh
631/tcp closed ipp
MAC Address: 00:E0:4C:4D:12:D9 (Realtek Semiconductor)

Nmap scan report for 10.0.0.5
Host is up (0.00019s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
2869/tcp open  icslap
3389/tcp open  ms-wbt-server
MAC Address: 00:E0:1C:3B:65:B3 (Cradlepoint)

Nmap done: 4 IP addresses (4 hosts up) scanned in 11.82 seconds
```

Figure 10. *Range of IP address*

see that in Figure 7 when we scan the system it shows a MAC address of the LAN Card. In Figure 8, you can see there are so many ports Opened and Closed but it could not be shown as a MAC address here. Next, we will scan Multiple IPs with the use of different shorthand notation.

Multiple IP Scanning

You can scan multiple IPs in different ways. The first is by providing full IP addresses as seen in Figure 9. You can also provide the range of IP addresses (see Figure 10), or by giving comma of every IP address (see Figure 11). The result is shown in the figures.

```
root@spider-desktop:/# nmap 10.0.0.2,4,5

Starting Nmap 6.00 ( http://nmap.org ) at 2012-08-02 13:03 IST
Nmap scan report for 10.0.0.2
Host is up (0.0027s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
5357/tcp open  wsdapi
MAC Address: 00:26:C6:15:23:FA (Intel Corporate)

Nmap scan report for 10.0.0.4
Host is up (0.00014s latency).
Not shown: 998 filtered ports
PORT    STATE  SERVICE
22/tcp  open   ssh
631/tcp closed ipp
MAC Address: 00:E0:4C:4D:12:D9 (Realtek Semiconductor)

Nmap scan report for 10.0.0.5
Host is up (0.00018s latency).
Not shown: 996 filtered ports
PORT     STATE SERVICE
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
2869/tcp open  icslap
3389/tcp open  ms-wbt-server
MAC Address: 00:E0:1C:3B:65:B3 (Cradlepoint)

Nmap done: 3 IP addresses (3 hosts up) scanned in 9.23 seconds
```

Figure 11. *Scanning by specific multiple IP*

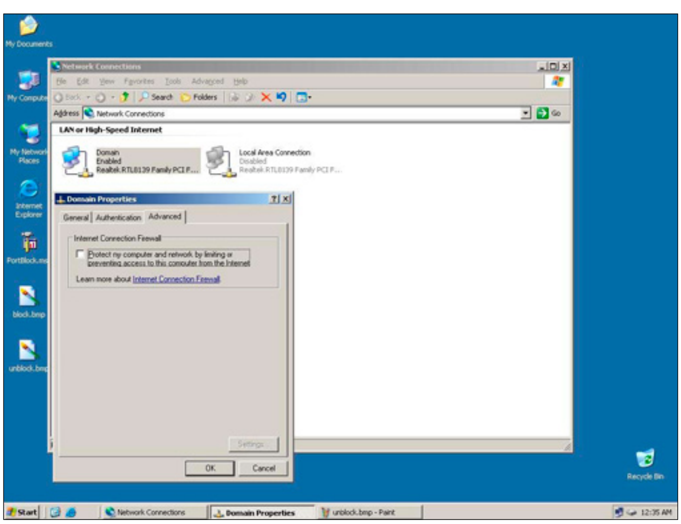


Figure 12. *Firewall is not on*

Host Discovery Scanning Technique / Ping Scanning Technique

Host Discovery OR Ping Scanning Technique is very useful. When we ping any host, we get information about whether the host system is live or not. In large organizations many administrators have blocked ICMP ping, so it's difficult to know if the system is live or not. Let's see an example. This is the 2003 Enterprise server. In this server if we do not start the firewall (you can see in Figure 12) then you get the pinging. So it's easy for us to find out whether the system is live or not. You can see the response of ping

```
H:\nmap-6.01>ping 192.168.1.100

Pinging 192.168.1.100 with 32 bytes of data:

Reply from 192.168.1.100: bytes=32 time=1ms TTL=128
Reply from 192.168.1.100: bytes=32 time=2ms TTL=128
Reply from 192.168.1.100: bytes=32 time=1ms TTL=128
Reply from 192.168.1.100: bytes=32 time=1ms TTL=128

Ping statistics for 192.168.1.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 2ms, Average = 1ms
```

Figure 13. *Getting the response*

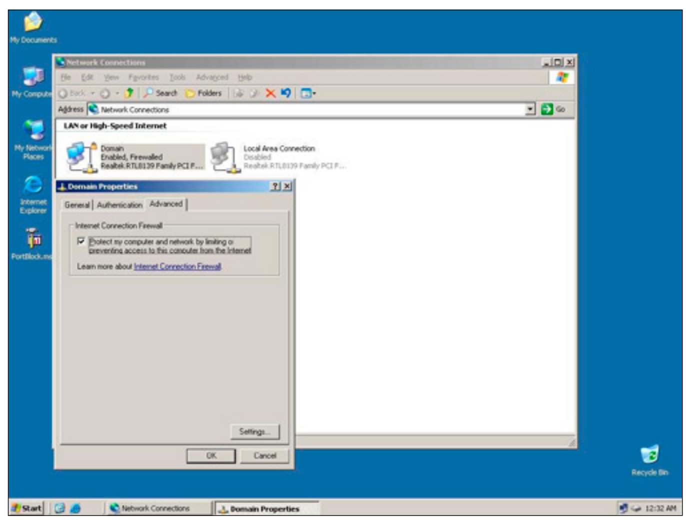


Figure 14. *Firewall is on*

```
H:\nmap-6.01>ping 192.168.1.100

Pinging 192.168.1.100 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.100:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Figure 15. *Not getting reply*

```
H:\nmap-6.01>nmap -sP 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-14 16:40 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (0.0020s latency).
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)
Nmap done: 1 IP address (1 host up) scanned in 2.30 seconds
```

Figure 16. *-sP Result*

replay in Figure 13. But if we activate the firewall then afterwards if we ping the system, it's very hard to find out whether the host is live or not. As per Figure 14, you can see if we activate the firewall after that we are unable to ping the system as we do not get any response of ICMP echo request see Figure 15.

Ping Scan

In this condition, it's hard to know if the host is up or not here. Nmap is performing an important role. If you want to ping only and know that the system is live, then use `-sP` command. Also refer to Figure 16.

Syntax

```
# nmap -sP <IP / Hostname > #nmap -sP 192.168.1.100
```

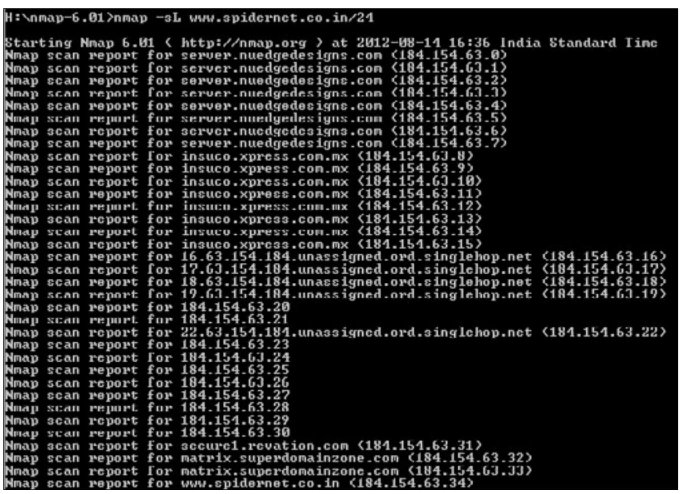
This option is also termed a Ping sweep. This is the most useful option for administrators if they want to check the network they use with CIDR also. This command is valuable because it's not going to do further query like Port Scanning, Service, OS detection, etc. It's also easy to use.

Host List Scanning

Syntax

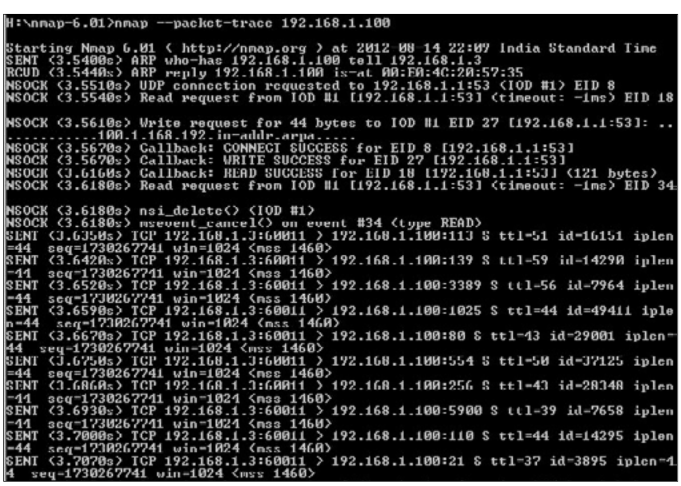
```
# nmap -sL <IP / Hostname / Domainname > #nmap
        -sL www.spidernet.co.in
```

In every Nmap's switch commands are easy to remember because of the short form (like -sL, which means scan List or List Scan). When you give the command `-sL` then you tell nmap to scan the reverse DNS lookup to the host / IP range / or from specific domain in the above Figure 17. You can see the spidernet.co.in in all lists of the NS Server. Really important information is revealed after the option `-sL`. You can find the purpose the IP address is used for and the location of the IP. When the command is executed nmap `-sL` that means it's not to



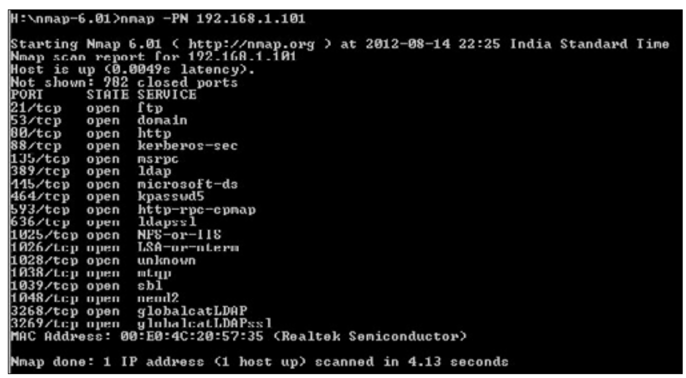
```
H:\nmap-6.01>nmap -sL www.spidernet.co.in/24

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-14 16:36 India Standard Time
Nmap scan report for server.nuedgedesigns.com (184.154.63.0)
Nmap scan report for server.nuedgedesigns.com (184.154.63.1)
Nmap scan report for server.nuedgedesigns.com (184.154.63.2)
Nmap scan report for server.nuedgedesigns.com (184.154.63.3)
Nmap scan report for server.nuedgedesigns.com (184.154.63.4)
Nmap scan report for server.nuedgedesigns.com (184.154.63.5)
Nmap scan report for server.nuedgedesigns.com (184.154.63.6)
Nmap scan report for server.nuedgedesigns.com (184.154.63.7)
Nmap scan report for insuco.xpress.com.mx (184.154.63.8)
Nmap scan report for insuco.xpress.com.mx (184.154.63.9)
Nmap scan report for insuco.xpress.com.mx (184.154.63.10)
Nmap scan report for insuco.xpress.com.mx (184.154.63.11)
Nmap scan report for insuco.xpress.com.mx (184.154.63.12)
Nmap scan report for insuco.xpress.com.mx (184.154.63.13)
Nmap scan report for insuco.xpress.com.mx (184.154.63.14)
Nmap scan report for insuco.xpress.com.mx (184.154.63.15)
Nmap scan report for 16.63.154.184.unassigned.ord.singlehop.net (184.154.63.16)
Nmap scan report for 17.63.154.184.unassigned.ord.singlehop.net (184.154.63.17)
Nmap scan report for 18.63.154.184.unassigned.ord.singlehop.net (184.154.63.18)
Nmap scan report for 19.63.154.184.unassigned.ord.singlehop.net (184.154.63.19)
Nmap scan report for 184.154.63.20
Nmap scan report for 184.154.63.21
Nmap scan report for 22.63.154.184.unassigned.ord.singlehop.net (184.154.63.22)
Nmap scan report for 184.154.63.23
Nmap scan report for 184.154.63.24
Nmap scan report for 184.154.63.25
Nmap scan report for 184.154.63.26
Nmap scan report for 184.154.63.27
Nmap scan report for 184.154.63.28
Nmap scan report for 184.154.63.29
Nmap scan report for 184.154.63.30
Nmap scan report for secure1.revation.com (184.154.63.31)
Nmap scan report for matrix.superdomainzone.com (184.154.63.32)
Nmap scan report for matrix.superdomainzone.com (184.154.63.33)
Nmap scan report for www.spidernet.co.in (184.154.63.34)
```

Figure 17. *-sL Result*



```
H:\nmap-6.01>nmap --packet-trace 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012 08 14 22:07 India Standard Time
SENT (3.5400s) ARP who-has 192.168.1.100 tell 192.168.1.3
RCVD (3.5440s) ARP reply 192.168.1.100 is-at 00:E0:4C:20:57:35
NSOCK (3.5510s) UDP connection requested to 192.168.1.1:53 (IOD #1) EID 8
NSOCK (3.5540s) Read request from IOD #1 [192.168.1.1:53] (timeout: -1ms) EID 18

NSOCK (3.5610s) Write request for 44 bytes to IOD #1 EID 27 [192.168.1.1:53]: ..
...........100.1.168.192.in-addr.arpa.....
NSOCK (3.5670s) Callback: CONNECT SUCCESS for EID 8 [192.168.1.1:53]
NSOCK (3.5670s) Callback: WRITE SUCCESS for EID 27 [192.168.1.1:53]
NSOCK (3.6160s) Callback: READ SUCCESS for EID 18 [192.168.1.1:53] (121 bytes)
NSOCK (3.6180s) Read request from IOD #1 [192.168.1.1:53] (timeout: -1ms) EID 34

NSOCK (3.6180s) nsi_delete() (IOD #1)
NSOCK (3.6180s) msevent_cancel() on event #34 (type READ)
SENT (3.6350s) TCP 192.168.1.3:60011 > 192.168.1.100:113 S ttl=51 id=16151 iplen
=44  seq=1730267741 win=1024 <mss 1460>
SENT (3.6420s) TCP 192.168.1.3:60011 > 192.168.1.100:139 S ttl=59 id=14290 iplen
=44  seq=1730267741 win=1024 <mss 1460>
SENT (3.6520s) TCP 192.168.1.3:60011 > 192.168.1.100:3389 S ttl=56 id=7964 iplen
=44  seq=1730267741 win=1024 <mss 1460>
SENT (3.6590s) TCP 192.168.1.3:60011 > 192.168.1.100:1025 S ttl=44 id=49411 iple
n=44  seq=1730267741 win=1024 <mss 1460>
SENT (3.6670s) TCP 192.168.1.3:60011 > 192.168.1.100:80 S ttl=43 id=29001 iplen=
44  seq=1730267741 win=1024 <mss 1460>
SENT (3.6750s) TCP 192.168.1.3:60011 > 192.168.1.100:554 S ttl=50 id=37125 iplen
=44  seq=1730267741 win=1024 <mss 1460>
SENT (3.6860s) TCP 192.168.1.3:60011 > 192.168.1.100:256 S ttl=43 id=28348 iplen
=44  seq=1730267741 win=1024 <mss 1460>
SENT (3.6930s) TCP 192.168.1.3:60011 > 192.168.1.100:5900 S ttl=39 id=7658 iplen
=44  seq=1730267741 win=1024 <mss 1460>
SENT (3.7000s) TCP 192.168.1.3:60011 > 192.168.1.100:110 S ttl=44 id=14295 iplen
=44  seq=1730267741 win=1024 <mss 1460>
SENT (3.7070s) TCP 192.168.1.3:60011 > 192.168.1.100:21 S ttl=37 id=3895 iplen=4
4  seq=1730267741 win=1024 <mss 1460>
```

Figure 18. *Packet Trac*



```
H:\nmap-6.01>nmap -PN 192.168.1.101

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-14 22:25 India Standard Time
Nmap scan report for 192.168.1.101
Host is up (0.0049s latency).
Not shown: 982 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
53/tcp    open  domain
80/tcp    open  http
88/tcp    open  kerberos-sec
135/tcp   open  msrpc
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
593/tcp   open  http-rpc-epmap
636/tcp   open  ldapssl
1025/tcp open  NFS-or-IIS
1026/tcp open  LSA-or-nterm
1028/tcp open  unknown
1038/tcp open  mtqp
1039/tcp open  sbl
1048/tcp open  neod2
3268/tcp open  globalcatLDAP
3269/tcp open  globalcatLDAPssl
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)

Nmap done: 1 IP address (1 host up) scanned in 4.13 seconds
```

Figure 19. *Result of -PN*

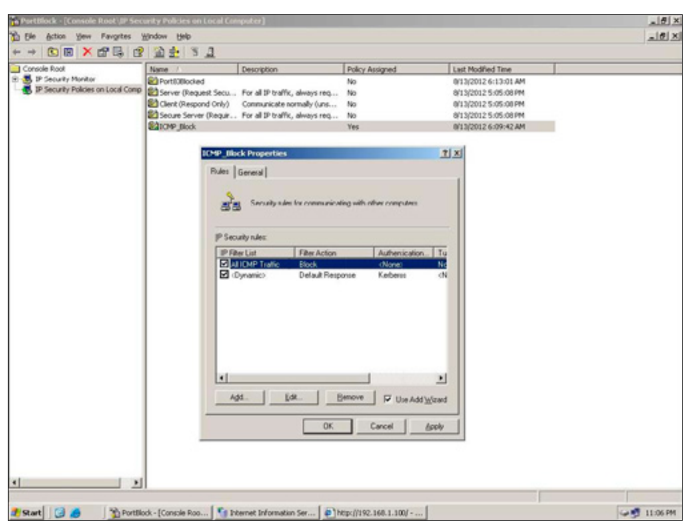


Figure 20. *ICMP Blocked by admin with IP sec*

send any packet to the target system. It works like a thief – it steals the information without an alert on the host IDS system and simply prints it.

Scanning Without Ping

When you ping the target host, our machine sends thousands of packets and also receives the thousands of packets (see Figure 18) to the system. This internal process is time consuming. This option is useful, for example, if the administrator knows the system is up in his list then there is no point to ping. If he uses the -PN option then he will get all of the ports' information and he will save time. This is also shown in Figure 19.

Syntax

```
# nmap -PN <IP / Hostname / Domainname > #nmap -PN www.
   spidernet.co.in
```

TCP SYN Ping

This ping is based on a particular port based ping. The option of `-PS` is used with any port. It is referred to as a TCP Syn ping because the SYN Flag is going to tell the target system that the connection establishment is in process. If the port is closed then the packet is sent back, but if the port is open, then it will proceed further. The Target system will send the ACK packet back to us, SYN will probe to the port 80, and a reply will be received from that port. You can see in Figure 20 that the 2003 server ICMP is blocked. In this situation, if we try to identify whether the machine is up or not and we ping the target Machine, then we get "Request time out" (see Figure 15). In this case if the ICMP is blocked but the WEB Server is running on PORT 80 and the site is up (see Figure 21), then our work will be easy. We send to Nmap the option -PS80 and we'll know whether the target host is available or not.

Syntax

```
# nmap -PS <Any Port> <IP / Hostname / Domainname > #nmap
   -PS80 192.168.1.100
```

Here we also use `-sP` for ping scan. Nmap gives so much flexibility in the use of different options simultaneously.

TCP ACK Ping

Similarly, TCP ACK Ping is also available in Nmap options. ACK ping is the same but there is a small difference between that and SYN ping.

Syntax

```
# nmap -PA <Any Port> <IP / Hostname / Domainname > #nmap
   -PA80 192.168.1.100
```

Nmap has these two options because there is a chance to bypass the firewall. If SYN ping does not work and admin blocks that, then ACK is useful in this case.

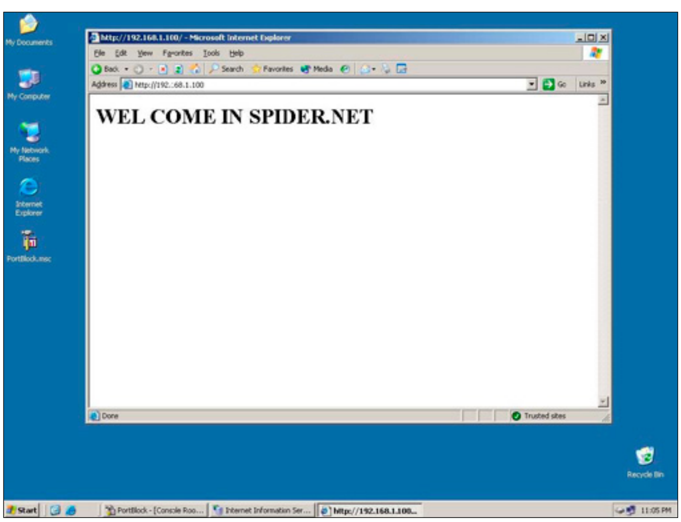


Figure 21. *Port 80 in Open Web server is Running*

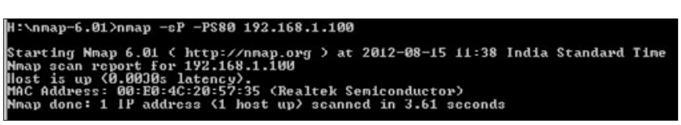


Figure 22. *Result of -PS*

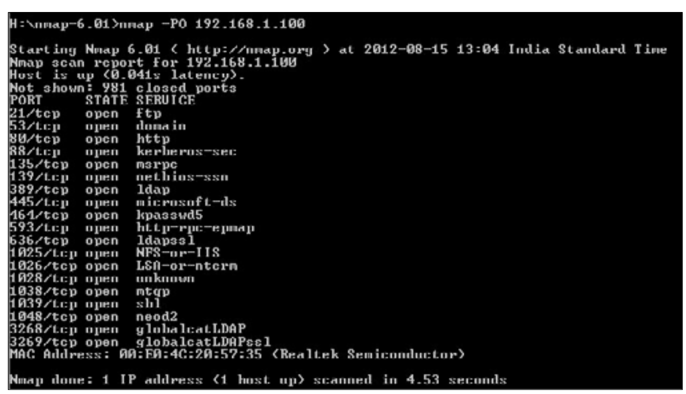


Figure 23. *Result of -PO*

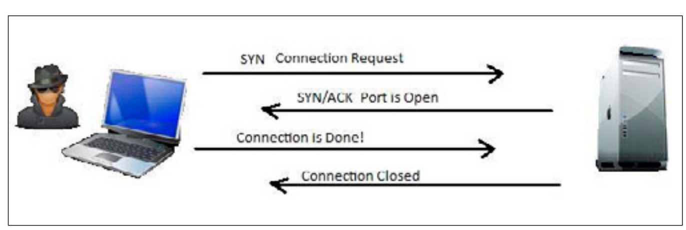


Figure 24. *Connection Establishment*

UDP Ping

UDP is a discovery option that sends the empty packet to the target host and admin only filters the TCP packet in the firewall. If it's poorly configured then you will get the response that it will allow you to get the information from the host. UDP ping uses the default probe port 31.338. You can also change this option in Nmap.

Syntax

```
# nmap -PU<Any Port> <IP / Hostname / Domainname > #nmap
   -PU 192.168.1.100
```

Three different ICMP Ping Scans

There are three different ICMP ping scans available in Nmap: 1) ICMP echo ping with option -PE; 2) ICMP Timestamp Ping with -PP; 3) ICMP Address Mask Ping -PM

1) ICMP echo ping -PE option is best in LAN and Internet by default. If you are not given any ping option, then -PE is applied.
2) ICMP Timestamp ping uses ICMP code 14. Some improperly configured systems may still reply to the ICMP timestamp.
3) ICMP address Mask ping uses ICMP code 18.

```
H:\nmap-6.01>nmap -sT 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 16:28 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (1.00s latency).
Not shown: 979 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
53/tcp    open  domain
80/tcp    open  http
88/tcp    open  kerberos-sec
110/tcp   open  pop3
111/tcp   open  rpcbind
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
593/tcp   open  http-rpc-epmap
636/tcp   open  ldapssl
1025/tcp open  NFS-or-IIS
1026/tcp open  LSA-or-nterm
1028/tcp open  unknown
1038/tcp open  mtqp
1039/tcp open  sbl
1048/tcp open  neod2
3268/tcp open  globalcatLDAP
3269/tcp open  globalcatLDAPssl
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)

Nmap done: 1 IP address (1 host up) scanned in 236.51 seconds
```

Figure 25. *-sT Connect Result*

```
H:\nmap-6.01>nmap -sU 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 17:00 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (0.0053s latency).
Not shown: 981 closed ports
PORT      STATE         SERVICE
53/udp    open          domain
88/udp    open|filtered kerberos-sec
123/udp   open          ntp
137/udp   open          netbios-ns
138/udp   open|filtered netbios-dgm
389/udp   open|filtered ldap
445/udp   open|filtered microsoft-ds
464/udp   open|filtered kpasswd5
500/udp   open|filtered isakmp
1030/udp open|filtered iad1
1032/udp open|filtered iad3
1035/udp open          mxxrlogin
1040/udp open|filtered netarx
1042/udp open|filtered afrog
1043/udp open|filtered boinc
1049/udp open|filtered td-postman
1457/udp open|filtered valisys-lm
3456/udp open|filtered IISrpc-or-vat
4500/udp open|filtered nat-t-ike
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)

Nmap done: 1 IP address (1 host up) scanned in 9.15 seconds
```

Figure 26. *-sU UDP Scan*

ICMP Echo Ping Syntax

```
# nmap -PE <IP / Hostname / Domainname >
#nmap -PE 192.168.1.100
```

ICMP Timestamp Syntax

```
# nmap -PP <IP / Hostname / Domainname >
#nmap -PP 192.168.1.100
```

ICMP Address mask Syntax

```
# nmap -PM <IP / Hostname / Domainname >
 #nmap -PM 192.168.1.100
```

IP Protocol Ping

Here you can see the tremendous flexibility of Nmap; -PO option is used for IP protocol scanning (for instance if you want to scan ICMP, IGMP, or other). The default is ICMP-1, IGMP-2 and IP in IP-4. (see Figure 23).

Syntax

```
# nmap -PO1,2,4 <IP / Hostname / Domainname >
#nmap -PO 192.168.1.100
```

Other Important option for Host Discovery technique

Nmap is really in-depth so it's not possible to see all the options in practice. Here, I'll show you some important Nmap switches. All of these options are used for host discovery techniques – you can use them as per your requirements:

--packet-trace

In Figure 18: Packet Tracing, you can find out how many packets are sent by nmap and received; you can even find the information about sequence number, Time to Live values, and TCP flag information.

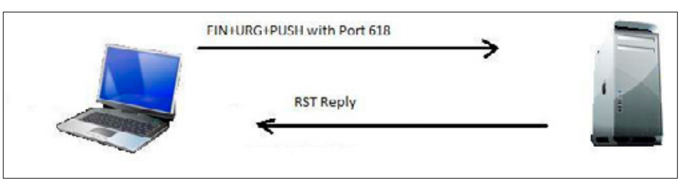


Figure 27. *Xmas scan*

```
H:\nmap-6.01>nmap -sX 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 18:47 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (0.0073s latency).
All 1000 scanned ports on 192.168.1.100 are closed
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)

Nmap done: 1 IP address (1 host up) scanned in 3.68 seconds
```

Figure 28. *Xmas scan Result*

--data-length <length>

If the Intrusion Detection System detects your scan, then you can also use this option because from `--data-length` switch you can control the length of bytes of data to every packet. This option also works with connectionless and connection-oriented protocols like TCP, UDP and ICMP also.

-n

-n option is used for disabling all DNS resolutions

-R

-R option enables all DNS Queries against the host. If the target host is down then it does not matter.

--dns-servers <dns server1> {, <server2>[,. . .]}

dns server1 – this is used for reverse query. This switch will directly go to the registry if the system is a Windows server, and if it's a Linux system, then it will try to read the resolve.conf file to obtain some important information about the dns server.

Advanced Scanning Techniques

TCP Connect Scan -sT

TCP Connect scan is an advanced scanning technique. First, it will request the target host for the connection sending by the SYN packet on any port like port 22, then, if the port is open, the host sends back an acknowledgment that it is open.

Again the system is going to connect with the target system, once the connection is finished then nmap `-sT` will start scanning the system. When all processes are done, the connection will be closed. In this technique there is also a drawback included. If the target system has an IDS, then it will catch you and generate a log after scanning, allowing the admin to easily see which IP scanned his system.

You can see the below Figure 24 to understand how the connection is established and closed. This is the disadvantage that they developed the TCP SYN / Stealth Scan `-sS` for. It's opposite the `-sT` option. You can see the result of `-sT` in Figure 25.

Syntax

```
# nmap -sT <IP / Hostname / Domainname > #nmap -sT
   192.168.1.100
```

TCP SYN Scan -sS

This type of scan needs a root privilege for the scanning. It's also called a stealthy scan because it does not need a full-fledged connection to the remote host. By default, it's a scan that is most common. Thousands of used TCP ports per second do not give any opportunity attention to the firewall.

Syntax

```
# nmap -sS <IP / Hostname / Domainname > #nmap -sS
   192.168.1.100
```

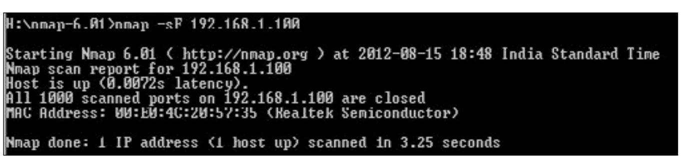
UDP Scan -sU

User Datagram Protocol (UDP) Services are scanned and enabled by the `-sU` switch. It is slower if we compare it with the TCP scan but it's more important because it's more complex than TCP. Many admins ignore this port because of its greater difficulty than TCP; it's a big mistake because some attackers are used to scanning this port which you can see below in Figure 26. Once we scan the 2003 server, we can see that ports 53, 123, and others are open.

Syntax

```
# nmap -sU <IP / Hostname / Domainname > #nmap -sU
   192.168.1.100
```

UDP sends an empty header to every port. UDP shows four states: Open, Open|filtered, Closed, and Filtered.

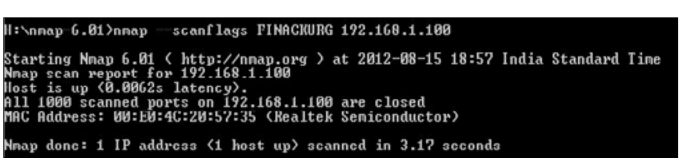


```
H:\nmap-6.01>nmap -sF 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 18:48 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (0.0072s latency).
All 1000 scanned ports on 192.168.1.100 are closed
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)

Nmap done: 1 IP address (1 host up) scanned in 3.25 seconds
```

Figure 29. *FIN scan Result*

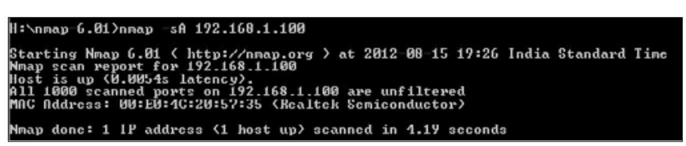
```
H:\nmap-6.01>nmap -sN 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 18:49 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (0.0063s latency).
All 1000 scanned ports on 192.168.1.100 are closed
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)

Nmap done: 1 IP address (1 host up) scanned in 3.27 seconds
```

Figure 30. *NULL scan Result*



```
H:\nmap-6.01>nmap --scanflags FINACKURG 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 18:57 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (0.0062s latency).
All 1000 scanned ports on 192.168.1.100 are closed
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)

Nmap done: 1 IP address (1 host up) scanned in 3.17 seconds
```

Figure 31. *Scanflags Scan Result*



```
H:\nmap-6.01>nmap -sA 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 19:26 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (0.0054s latency).
All 1000 scanned ports on 192.168.1.100 are unfiltered
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)

Nmap done: 1 IP address (1 host up) scanned in 4.19 seconds
```

Figure 32. *FIN scan Result*

All are different. Open means any UDP is responding from host. Open|filtered means you can't get any response, even retransmission. Closed means the ICMP echo request is unreachable. Filtered means the ICMP is unreachable with different code and type. By default, the UDP scan is slow but if you want to speed the UDP scan then you have to put in a different option with `-sU`. You can also control the slow host by putting `-host-timeout` option, `-v` option for the enabled verbosity mode, etc.

TCP Xmas, Null, and Fin Scans with --scanflags

Before we understand Xmas, Null and Fin Scan, we need to know what happens when a connection is established with SYN, FIN, ACK, URG, PUSH and RESET flag. SYN and Fin Flags are used for connection establishment and close the TCP Connection. ACK flag is set so that the acknowledgment field is valid, and gets the attention from the target system. The URG flag narrates the Segment containing urgent data, while the PUSH flag terms as a sender invoke the push operation, which indicates to the receiving side of TCP that it should notify the receiving process of this fact. Finally, the RESET flag is denoted, as the receiver has become confused and wants to abort the connection. Now, let's see what the Xmas Scan can do. This scan is turned On or Off by sending bytes much like the Christmas tree. A closed port is a response to an Xmas tree scan with RST as you can see in Figure 27.

Syntax

```
# nmap -sX <IP / Hostname / Domainname > #nmap -sX
   192.168.1.100
```

TCP Fin Scan

In this scan, TCP Fin bit is active when packets are sent in an attempt to solicit a TCP ACK from the destination target host. This is another choice for Scanning and gathering information from the Target system which is protected by Firewall.

Syntax

```
# nmap -sF <IP / Hostname / Domainname > #nmap -sF
   192.168.1.100
```

Figure 33. *-sO Scan Result*

TCP Null Scan

TCP Null scanning is fast compared to other Port scanning options. From this scan, the TCP flags are enabled and you can find that the packet header is 0. If the Port is closed on the target machine then the Null scan will not send flags in the packet header. Its reply will be by the RST Packets. This type of scanning has a major advantage of scanning through stateless firewalls or ACL filters.

Syntax

```
# nmap -sN <IP / Hostname / Domainname> #nmap -sN
   192.168.1.100
```

You can find similarity in all Figures of the TCP scan FIN, Null and Xmas observed in Figures 28, 29, 30 as you see that the result is the same. You can customize these three scans with the -scanflags. This option provides a lot of flexibility in scanning.

Syntax

```
# nmap --scanflags FINACKURGPSH <IP / Hostname /
Domainname> #nmap --scanflags FINACKURGPSH 192.168.1.100
```

TCP ACK Scan

First we have to understand the result that the ACK scan gives. Unfiltered -(TCP RST response) means special rules apply on the target's firewall. Filtered -(ICMP unreachable error OR No response) means the system is protected by the firewall. You can see in Figure 32 that "All 1000 scanned ports on 192.168.1.100 are unfiltered."

Syntax

```
# nmap -sA <IP / Hostname / Domainname > #nmap -sA 192.168.1.100
```

Figure 34. *--O Scan Result*

Other Important options for Advanced Scanning techniques

In advanced scanning there are so many options that are available but we will not cover them all.

--send-eth

This option tells Nmap to bypass the IP layer on your system and send raw Ethernet packets on the data link layer. It's a rarely used option.

Syntax

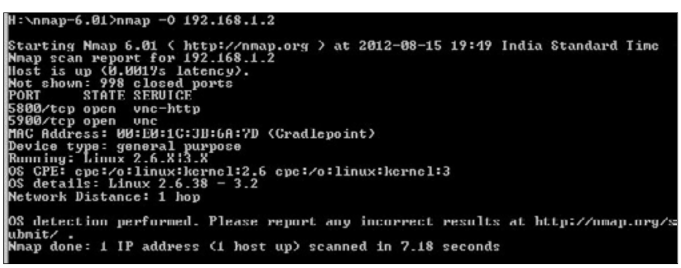
```
# nmap -send-eth <IP / Hostname / Domainname> #nmap -send-
   eth 192.168.1.100
```

-sO

This option is used for Scanning Protocol. From this scan you know which protocol is running on the target host. The most common protocol is TCP, UDP and ICMP. You can see Figure 33 while the 2003 server is scanning.

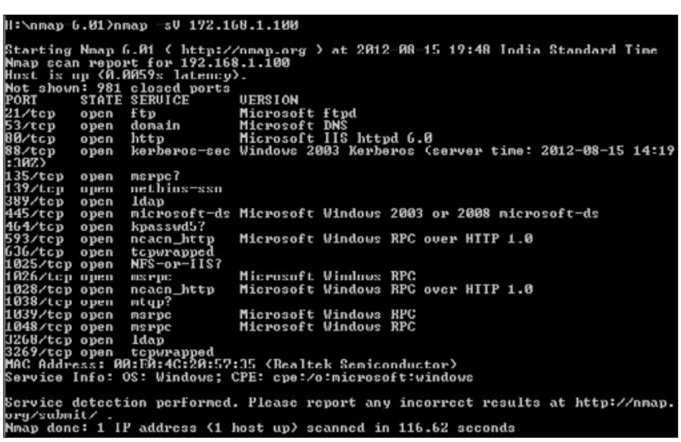
send-ip

These options forcefully tell Nmap to scan using the local system's IP stack instead of generating raw Ethernet packets. It is used in rare cases.



```
H:\nmap-6.01>nmap -O 192.168.1.2

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 19:49 India Standard Time
Nmap scan report for 192.168.1.2
Host is up (0.0017s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
5800/tcp open  vnc-http
5900/tcp open  vnc
MAC Address: 00:E0:1C:3D:6A:7D (Cradlepoint)
Device type: general purpose
Running: Linux 2.6.X|3.X
OS CPE: cpe:/o:linux:kernel:2.6 cpe:/o:linux:kernel:3
OS details: Linux 2.6.38 - 3.2
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at http://nmap.org/s
ubmit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.18 seconds
```

Figure35. *-sV Scan Result*



```
H:\nmap-6.01>nmap -sV 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 19:48 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (0.0059s latency).
Not shown: 981 closed ports
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          Microsoft ftpd
53/tcp    open  domain       Microsoft DNS
80/tcp    open  http         Microsoft IIS httpd 6.0
88/tcp    open  kerberos-sec Windows 2003 Kerberos (server time: 2012-08-15 14:19
:30Z)
135/tcp   open  msrpc?
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds Microsoft Windows 2003 or 2008 microsoft-ds
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http   Microsoft Windows RPC over HTTP 1.0
636/tcp   open  tcpwrapped
1025/tcp  open  NFS-or-IIS?
1026/tcp  open  msrpc        Microsoft Windows RPC
1028/tcp  open  ncacn_http   Microsoft Windows RPC over HTTP 1.0
1038/tcp  open  mtqp?
1039/tcp  open  msrpc        Microsoft Windows RPC
1048/tcp  open  msrpc        Microsoft Windows RPC
3268/tcp  open  ldap
3269/tcp  open  tcpwrapped
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at http://nmap.
org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 116.62 seconds
```

Figure 36. *-sV Scan Result*

Name Based Port Scanning

Multiple use of -p

-p option has multiple uses because you can scan based on the service name like smtp, pop2 etc, and you can also scan on the port number like 53, 25 etc. This is the most flexible option ever because if you want to scan with the UDP or TCP port, then you have to simply define the U:[Port number] or T:[Port number]. You can also use the wildcard with -p "*". This tells Nmap to scan all ports.

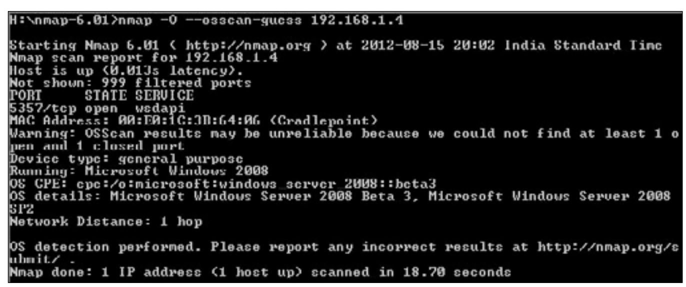
Syntax

```
# nmap -p [port number with comma or range] <IP / Hostname
   / Domainname>
#nmap -p 25,80,53-200 192.168.1.100
# nmap -p [name] <IP / Hostname / Domainname>
#nmap -p smtp,http 192.168.1.100
# nmap -p U: [port number] T: [port number] <IP
/ Hostname / Domainname> #nmap -p U:53,T:25 192.168.1.100
# nmap -p "*" <IP / Hostname / Domainname> #nmap -p "*"
   192.168.1.100
```

OS & Service Scanning

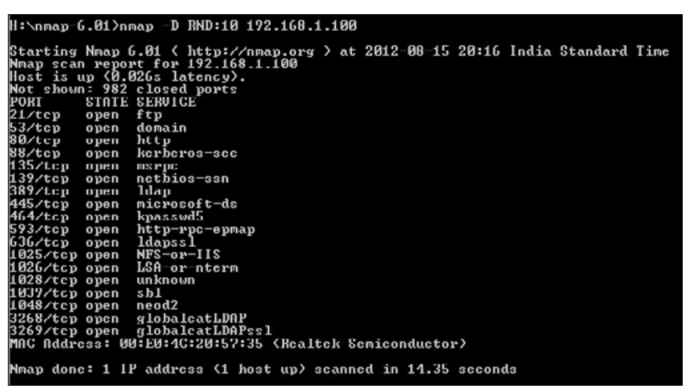
Operating System Detection

For OS detection mostly one port is open or one port is closed. `-O` option is used for knowing which operating system is running on the target system. You can see in Figure 34. This is the Windows 2003 server and in Figure 35 Ubuntu is installed.



```
H:\nmap-6.01>nmap -O --osscan-guess 192.168.1.4

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 20:02 India Standard Time
Nmap scan report for 192.168.1.4
Host is up (0.013s latency).
Not shown: 999 filtered ports
PORT     STATE SERVICE
5357/tcp open  wsdapi
MAC Address: 00:E0:1C:3D:64:06 (Cradlepoint)
Warning: OSScan results may be unreliable because we could not find at least 1 o
pen and 1 closed port
Device type: general purpose
Running: Microsoft Windows 2008
OS CPE: cpe:/o:microsoft:windows_server_2008::beta3
OS details: Microsoft Windows Server 2008 Beta 3, Microsoft Windows Server 2008
SP2
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at http://nmap.org/s
ubmit/ .
Nmap done: 1 IP address (1 host up) scanned in 18.70 seconds
```

Figure 37. *-osscan-guess Scan Result*



```
H:\nmap-6.01>nmap -D RND:10 192.168.1.100

Starting Nmap 6.01 ( http://nmap.org ) at 2012-08-15 20:16 India Standard Time
Nmap scan report for 192.168.1.100
Host is up (0.026s latency).
Not shown: 982 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
53/tcp   open  domain
80/tcp   open  http
88/tcp   open  kerberos-sec
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
389/tcp  open  ldap
445/tcp  open  microsoft-ds
464/tcp  open  kpasswd5
593/tcp  open  http-rpc-epmap
636/tcp  open  ldapssl
1025/tcp open  NFS-or-IIS
1026/tcp open  LSA-or-nterm
1028/tcp open  unknown
1037/tcp open  sbl
1048/tcp open  neod2
3268/tcp open  globalcatLDAP
3269/tcp open  globalcatLDAPssl
MAC Address: 00:E0:4C:20:57:35 (Realtek Semiconductor)

Nmap done: 1 IP address (1 host up) scanned in 14.35 seconds
```

Figure 38. *Decoy Result*

Syntax

```
# nmap -O <IP / Hostname / Domainname> #nmap -O
   192.168.1.100
```

Service Detection

Service Detection option is used as the -sV option. From this option you can find which service is running on the target host.

Syntax

```
# nmap -sV <IP / Hostname / Domainname> #nmap -sV
   192.168.1.100
```

Guess Unknown OS

This scan shows you the possible matches for the target OS system. For this scan, you can use -osscan-guess option.

Syntax

```
# nmap -osscan-guess <IP / Hostname / Domainname>
#nmap -osscan-guess 192.168.1.100
```

Firewall Evasion Technique

Spoof MAC address

In this example, you can see that Nmap generates a fake MAC address used for scanning. There are three options for spoofing MAC addresses. The first one is to give 0; -nmap will then generate random MAC addresses of any company like 3com or other. You can even specify the MAC Address, and you can give the Vendor name also.

Figure 39. *-smb-os-discovery*

Syntax

```
# nmap -spoof-mac [vendor | MAC | 0] <IP
/ Hostname / Domainname> #nmap -spoof-mac 0 192.168.1.100
```

Decoy Use

Decoy option gives the best performance during scanning because it generates additional packets and creates a virtualization that the system is scanned by multiple systems. From this option, it is hard to trace which system is scanning OR where the scanning is coming from. You can specify the decoys like decoys1, decoys2, etc., see Figure 38.

Syntax

```
# nmap -D RND:Number of Decoy <IP / Hostname
/ Domainname> #nmap -D RND:10 192.168.1.100
```

Nmap Scripting Engine

nmap --script smb-os-discovery 192.168.1.100 + smb-os-discovery gives you the result (which OS is running on the target system).

Syntax

```
# nmap --script smb-os-discovery <IP / Hostname
/ Domainname> #nmap --script smb-os-discovery
   192.168.1.100~
```

Figure 39: -smb-os-discovery 2) *nmap --script smb-system-info 192.168.1.100* --script smb-system-info is giving the information about the system.

Syntax

```
# nmap --script smb-system-info <IP / Hostname
/ Domainname> # nmap --script smb-system-info
   192.168.1.100~
```

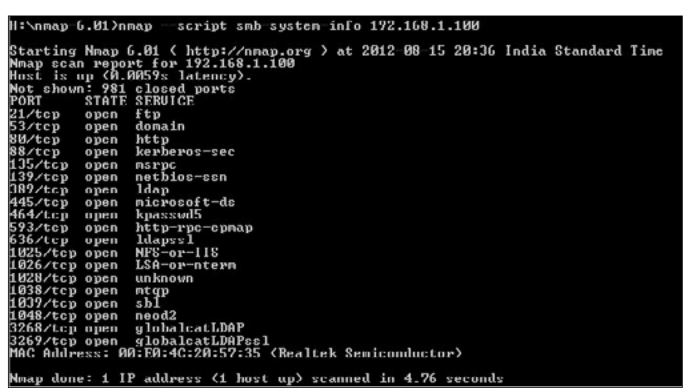


Figure 40. *-smb-system-info*

Nmap is very complex. There is also a time based scanning technique that is available and an Nmap Scripting Engine which is a very useful option in Nmap. Using this option, you can find all the information on users, shares, etc. NSE scripts define a list of categories they belong to. Currently defined categories are auth, broadcast, default, discovery, dos, exploit, external, fuzzer, intrusive, malware, safe, version, and vuln. Category names are not case-sensitive, NSE scripts consist of a handful of descriptive fields, a rule defining when the script should be executed, and an action function containing the actual script instructions. Values can be assigned to the descriptive fields just as you would assign any other Lua variables.

SAHIL KHAN

SAHIL KHAN aka "Makbulkhan" is the founder and director of Spider.net Institute. Sahil lives in Palanpur, North Gujarat. He states that teaching and management are his passions, and that he is addicted to computers. He started his career in 1998 in APTECH and Dollhans Company. After giving up his dream of becoming a Team Leader and Sr. System Administrator, he has committed himself to doing something for his motherland -specifically, opening up a Computer Hardware Networking Institute which shapes students into engineers, who will work for top international technological companies.

How to Use The Mac OS X Hackers Toolbox

When you think of an operating system to run pen testing tools on, you probably think of Linux and more specifically, BackTrack Linux. BackTrack Linux is a great option and one of the most common platforms for running pen testing tools. If you are a Mac user, then you would most likely run a virtual machine of BackTrack Linux.

While this is a great option, sometimes it is nice to have your tools running on the native operating system of your computer. Another benefit is not having to share your system resources with a virtual machine. This also eliminates the need to transfer files between your operating system and a virtual machine, and the hassles of having to deal with a virtual machine. Also by running the tools within OS X, you will be able to seamlessly access all of your Mac OS X applications.

My attack laptop happens to be a MacBook Pro and I started out running VirtualBox with a BackTrack Linux virtual machine. I recently started installing my hacking tools on my MacBook Pro. I wanted to expand the toolset of my Mac, so I started with Nessus, nmap, SQLMap, and then I installed Metasploit. My goal is to get most, if not all, of the tools I use installed on my MacBook Pro and run them natively within OS X. Since Mac OS X is a UNIX based operating system, you get great tools that come natively with UNIX operating systems such as netcat and SSH. You also have powerful scripting languages installed such as Perl and Python. With all of the benefits and features of the Mac OS X, there is no reason to not use Mac OS X for your pen testing platform. I was really surprised to see that there's not a lot of information on the subject of using Mac OS X as a pen testing/hacking platform. Metasploit was the toughest application to get running on Mac OS X and that was mostly due to the PostgreSQL database setup. The majority of hacking tools are command line based, so they are easy and fairly straightforward to install.

In this article, I am going to take you through the installation and configuration of some of the most popular and useful hacking tools, such as Metasploit, on Mac OS X. If you are interested in maximizing the use of your Mac for pen testing and running your tools natively, then you should find this article helpful.

The Tools

The pen test tools we will be installing are must-haves and all of them are free, with the exception of Burp Suite and Nessus (although Burp Suite has a free version, which offers a portion of the Burp Suite tools for free). The tools offered for free with Burp Suite are useful tools and I highly recommend them. The professional version of Burp Suite is reasonably priced.

- Metasploit Framework
- Nmap
- SQLmap
- Burp Suite
- Nessus

- SSLScan
- Wireshark
- TCPDUMP
- Netcat

Metasploit Framework

The Metasploit Framework is one of the most popular and powerful exploit tools for pen testers and a must have for pen testers. The Metasploit Framework simplifies the exploitation process and allows you to manage your pen tests with the workspace function in Metasploit. Metasploit also allows you to run nmap within Metasploit and the scan information is organized by project with the workspace function. You can create your own exploits and modify existing exploits in Metasploit. Metasploit has too many features to mention in this article, and the scope of this article is to demonstrate how to install Metasploit and other pen testing tools.

The Install

Before we install Metasploit, we need to install some software dependencies. It is a little more work to install Metasploit on Mac OS X, but it will be worth it. Listed below are the prerequisite software packages.

Software Prerequisites

- MacPorts
- Ruby1.9.3
- Homebrew
- PostgreSQL

MacPorts Installation

Install Xcode

- Xcode Install from the Apple App Store, or it can be downloaded from the following URL; *https://developer.apple.com/xcode/*
- Once Xcode is installed, go into the Xcode preferences and install the "Command Line Tools". (see Figure 1)

Install the MacPorts app

- Download and install the package file (.dmg) file from the MacPorts web site; *https://distfiles.macports.org/MacPorts/*
  Once the files are downloaded, install MacPorts.
  More information on MacPorts can be found here: *http://www.macports.org/install.php*
- Run MacPorts selfupdate to make sure it is using the latest version.
  From a terminal window run the following command:

```
$ sudo port selfupdate
```

Ruby 1.9.3

Mac OS X is preinstalled with Ruby, but we want to upgrade to Ruby 1.9.3

- We will be using MacPorts to upgrade Ruby.
  From a terminal window run the following command:

```
$ sudo port install ruby19 +nosuffix
```

- The default Ruby install path for MacPorts is: `/opt/local/`
  It's a good idea to verify that the PATH is correct, so that `opt/local/bin` is listed before `/usr/bin`. You should get back something that looks like this:

```
/opt/local/bin:/opt/local/sbin:/usr/bin:/bin:/usr/
sbin:/sbin
```

You can verify the path by entering the following syntax in a terminal window:

```
$ echo $PATH
```

To verify the Ruby install locations, enter this syntax:

```
$ which ruby gem
```

You should get back the following response:

```
/opt/local/bin/ruby
/opt/local/bin/gem
```

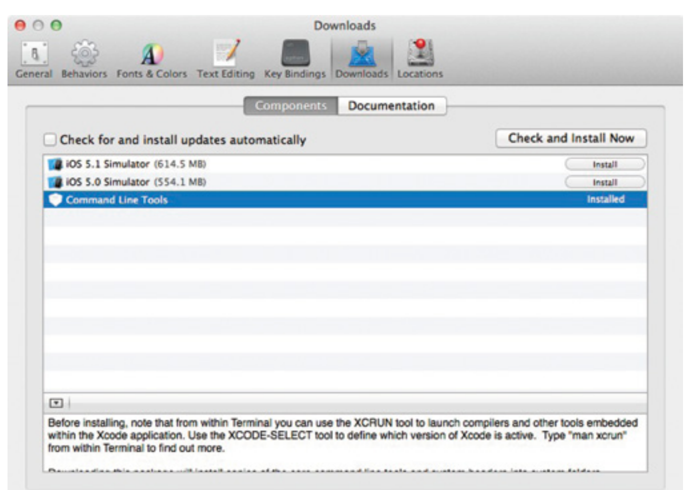


Figure 1. *Install "Command Line Tools"*

Database Installation

A database is not required to run, but some of the features of Metasploit require that you install a database. The workspace feature of Metasploit is one of the really nice features of Metasploit that requires a database. Workspace allows easy project organization by offering separate workspaces for each project. PostgreSQL is the vendor recommended and supported database, but MySQL can be used. In this article, we will be using PostgreSQL.

We will use Homebrew to install PostgreSQL. I tried a few different installation methods, but this is the easiest way to install PostgreSQL. Homebrew is a good method to install Open Source software packages.

- First we will install Homebrew.
  From a terminal window run the following command:

```
$ ruby -e “$(curl -fsSkL raw.github.com/mxcl/homebrew/go)”
```

- Next we will install PostgreSQL using Homebrew.
  From a terminal window run the following command:

```
$ brew install postgresql
```

Figure 2. *This is one of the many Metasploit screens you will see when launching Metasploit*

- Next we initialize the database, configure the startup, and start PostgreSQL. From a terminal window run the following command:

```
initdb /usr/local/var/postgres cp /usr/
local/Cellar/postgresql/9.1.4/homebrew.mxcl.
postgresql.plist ~/Library/LaunchAgents/
launchctl load -w ~/Library/LaunchAgents/
homebrew.mxcl.postgresql.plist pg_ctl -D /
usr/local/var/postgres -l /usr/local/var/
postgres/server.log start
```

- Database configuration
  In this step we will create our Metasploit database and the database user.
  - The Homebrew install does not create the postgres user, so we need to create the postgres user to create databases and database users.
    At a command prompt, type the following:

```
$ createuser postgres_user -P
$ Enter password for new role: password
$ Enter it again: password
$ Shall the new role be a superuser? (y/n) y
$ Shall the new role be allowed to create databases? (y/n) y
$ Shall the new role be allowed to create more new roles?
  (y/n) y
```

  - Creating the database user
    At a command prompt, type the following:

```
$ createuser msf_user -P
$ Enter password for new role: password
$ Enter it again: password
$ Shall the new role be a superuser? (y/n) n
$ Shall the new role be allowed to create databases? (y/n) n
$ Shall the new role be allowed to create more new roles?
  (y/n) n
```

  - Creating the database
    At a command prompt, type the following:

```
$ createdb --owner=msf_user msf_database
```

  - Install the pg gem.
    At a command prompt, type the following:

```
$ gem install pg
```

The database and database user are created, so now it is time to install Metasploit.

Metasploit software installation

The dependencies have been installed and next we will install the Metasploit software.

- Download the Metasploit source code for installation using the link provided below and do not download the .run file from the Metasploit download page. Download the Metasploit tar file from: *http://downloads.metasploit.com/data/releases/framework-latest.tar.bz2*.
- Once the download is complete, untar the file. If you have software installed to unzip or untar files, then it should untar the file when the file is finished downloading. I use StuffIt Expander and it untarred the file for me upon completion of the download. If you need to manually untar the file, type this command at the command line and it will untar the file into the desired directory:

```
$ sudo tar -xvf framework-lastest-tar.bz2 -C /opt
```

  If the file was untarred for you as mentioned, you will need to move the Metasploit source file structure to the opt directory. Your directory structure should look like this:

```
/opt/metasploit3/msf3
```

Starting Metasploit

Now that Metasploit is installed, we will start Metasploit for the first time. You will need to navigate to the Metasploit directory and start Metasploit.

- Navigate to the Metasploit directory with the following syntax entered at the command line:

```
$ cd /opt/metasploit/msf3
```

- To start Metasploit, simply enter the following syntax:

```
$ sudo ./msfconsole
```

  You will get one of the many Metasploit screens like the one in Figure 2.

Connecting to the database

In this next step, we will connect Metasploit to our PostgreSQL database. From the Metasploit prompt, type the following syntax:

```
msf > db_connect msf_user:password@127.0.0.1/msf_database
```

You will see the following message and you should be connected.

Listing 1. *Database Backend Commands as displayed in the Metasploit console*

```
Database Backend Commands
=========================

Command           Description
-------           -----------
creds             List all credentials in the database
db_connect        Connect to an existing database
db_disconnect     Disconnect from the current database instance
db_export         Export a file containing the contents of the database
db_import        Import a scan result file (filetype will be auto-detected)
db_nmap           Executes nmap and records the output automatically
db_rebuild_cache  Rebuilds the database-stored module cache
db_status         Show the current database status
hosts             List all hosts in the database
loot              List all loot in the database
notes             List all notes in the database
services          List all services in the database
vulns             List all vulnerabilities in the database
workspace         Switch between database workspaces
```

```
[*] Rebuilding the module cache in the background...
```

Type in the following syntax to verify the database is connected:

```
msf > db_status
```

You will get the following back verifying the database is connected:

```
[*] postgresql connected to msf_database
```

The database is now connected to Metasploit, but once you exit Metasploit the database will be disconnected. To configure Metasploit to automatically connect on startup, we will have to create the msfconsole.rc file.

Enter the following syntax at the command prompt:

```
$ cat > ~/.msf3/msfconsole.rc << EOF db_connect
-y /opt/metasploit3/config/database.yml
EOF
```

Updating Metasploit

Now that we have Metasploit installed and configured, we will update the Metasploit installation. From the command prompt, type the following syntax:

```
$ ./msfupdate
```

This can take a while, so just sit back and let the update complete. Make sure to update Metasploit frequently so you have the latest exploits.

The benefits of Metasploit with database

Now that Metasploit is installed, the database is connected and ready to use. So what can you do with Metasploit with a database that you couldn't do without one? Below is a list of new Metasploit Database Backend Commands taken directly from the Metasploit console. The commands are pretty much self-explanatory, but it should be noted that db_import allows you to import nmap scans done outside of Metasploit. This comes in handy when you are working with others on a pen test and you want to centrally manage your pen test data. As mentioned earlier, workspace helps you manage your pen tests by allowing you to store them in separate areas of the database. A great reference guide for Metasploit can be found at Offensive Security's website: *http://www.offensive-security.com/metasploit-unleashed/Main_Page*.

Nmap

Nmap is an open source network discovery and security auditing tool. You can run nmap within Metasploit, but it is good to have nmap installed so you can run nmap outside of Metasploit. We will use Homebrew to install nmap. From the command prompt, type the following syntax:

```
$ brew install nmap
```

Visit the Nmap website for the Nmap reference guide: *http://nmap.org/book/man.html*.

SQLmap

SQLmap is a penetration testing tool that detects SQL injection flaws and automates SQL injection. From the command prompt, type the following syntax:

```
$ git clone https://github.com/sqlmapproject/sqlmap.git
   sqlmap-dev
```

Burp Suite

Burp Suite is a set of web security testing tools, including Burp Proxy. To install Burp Suite, download it from: *http://www.portswigger.net/burp/download.html.*

To run Burp, type the following syntax from the command prompt:

```
$ java -jar -Xmx1024m burpsuite_v1.4.01.jar
```

For more information on using Burp, go to the Burp Suite website: *http://www.portswigger.net/burp/help/*.

Nessus

Nessus is a commercial vulnerability scanner and it can be downloaded from the Tenable Network website: *http://www.tenable.com/products/nessus/nessus-download-agreement*.

Download the file Nessus-5.x.x.dmg.gz, and then double click on it to unzip it. Double click on the Nessus-5.x.x.dmg file, which will mount the disk image and make it appear under "Devices" in "Finder". Once the volume "Nessus 5" appears in "Finder", double click on the file Nessus 5.

The Nessus installer is GUI based like other Mac OS X applications, so there are no special instructions to document. The Nessus 5.0 Installation and Configuration Guide as well as the Nessus 5.0 User Guide can be downloaded from the documentation section of the Tenable Network website: *http://www.tenable.com/products/nessus/documentation*.

SSLScan

SSLScan queries SSL services, such as HTTPS, in order to determine the ciphers that are supported.

To install sslscan, type the following syntax at the command prompt:

```
$ brew install sslscan
```

Wireshark

Wireshark is a packet analyzer and can be useful in pen tests. Wireshark DMG package can be downloaded from the Wireshark website: *http://www.wireshark.org/download.html*. Once the file is downloaded, double click to install Wireshark.

TCPDUMP

TCPDUMP is a command line packet analyzer that is preinstalled on Mac OS X. For more information consult the man page for tcpdump by typing the following syntax at the command prompt:

```
$ man tcpdump
```

Netcat

Netcat is a multipurpose network utility that is preinstalled on Mac OS X. Netcat can be used for port redirection, tunneling, and port scanning to name just a few of the capabilities of Netcat. Netcat is used a lot for reverse shells. For more information on Netcat, type the following syntax at the command prompt:

```
$ man nc
```

Conclusion

By following the instructions in this article, you will have a fully functional set of hacking tools installed on your Mac and you will be able to run them natively without having to start a virtual machine or deal with the added administrative overhead that comes with running a virtual machine. You will also not have to share resources with a virtual machine. I hope you found this article useful and I hope you enjoy setting up your Mac OS X hacker toolbox as much as I did. With Macs increasing in popularity, I can only imagine that they will become more widely used in pen testing.

PHILLIP WYLIE

Phillip Wylie is a security consultant specializing in penetration testing, network vulnerability assessments and application vulnerability assessments. Phillip has over 8 years of experience in information security and 7 years of system administration experience.

Basic Unix Queuing Techniques

It occasionally happens that our incoming or outgoing data cannot be processed as it is generated or, for some reason, we choose to process it at a later time.

A typical example might be a client-server system, where it is necessary to queue the socket descriptors of incoming connections because of some limit on the number of active processes, or a message hub, which accepts data synchronously, but must rely on other processes to remove the data asynchronously. Apart from the numerous commercially-available third party implementations of queuing systems, Unix has two highly efficient queuing mechanisms, which can be used for extremely low overhead systems of queues.

Kernel mode queues

The kernel uses queues internally for the implementation of functions such as device drivers, and the system call interface to this mechanism is available for the implementation of application programs. The queues so produced are implemented in memory, so they are very fast. However, because there is no permanent storage of the data, these queues are also non-persistent. This means that if the process or the machine crashes, all of the queued data will be lost, and all incoming data will never be enqueued.

User mode queues

In this section, we will concentrate on disk-based user mode queues. The kernel mode queuing system, which will be covered in an upcoming Advanced Queuing Article, is a bit limited, and it is sometimes more convenient to use the user mode queue library functions which offer a little more functionality, namely:

- Notification of message arrival, by sending a signal to the monitoring process.
- Prioritization of messages

There are only four fundamental commands to remember:

- `mq_open()` – opens an existing queue, or creates a new queue
- `mq_send()` – enqueues a message
- `mq_receive()` – dequeues a message
- `mq_notify()` – notifies a process of the arrival of a message

The remaining five commands perform housekeeping tasks:

- `mq_close()` – closes a queue
- `mq_unlink()` – deletes a queue from the disk
- `mq_getattr()` – interrogates a queue's characteristics
- `mq_setattr()` – sets a queue's characteristics

A single structure definition is used to set and get the queue's attributes, and is defined as:

```
struct mq_attr {
    long    mq_flags      /* message queue flags */
    long    mq_maxmsg     /* maximum number of messages */
    long    mq_msgsize    /* maximum message size */
    long    mq_curmsgs    /*  number of messages currently
   queued */
};
```

The mq series of commands all relate to disk based queues. The queues themselves are created in the `/tmp` directory and are always referred to in the commands, as if they were situated below the root directory.

Thus to create a queue called 'zq', we would call `mq_open()`, like this:

```
Int qd;
struct mq_attr atr;

    atr.mq_maxmsg = 100;
    atr.mq_msgsize = 255;

    if((qd = mq_open("/zq", O_RDWR|O_CREAT, 0755, &atr))
   == (mqd_t)-1){
        perror("mq_open");
    }
```

Notice the similarity between the above syntax, and that of the `open()` command, for a file. The returned value is the queue descriptor, while the flags are exactly the same, as defined in fcntl.h for those relating to a file. The pointer to the 'atr' structure permits the setting of the maximum number of messages, and the maximum message size, prior to calling `mq_open`.

Enqueuing a message is analogous to a `write()` on a file:

```
char *msg = "xyz";
int priority = 5;

    if(mq_send(qd, msg, strlen(msg), priority) == -1){
        perror("mq_send");
    }
```

The extra parameter, 'priority' determines the order that the message will be removed from the queue when it is dequeued, with '1' being the highest priority.

The dequeuing is performed by `mq_receive()`:

```
unsigned char data[8192];
int priority;
int n;

    If((n = mq_receive(qd, (char *)data, sizeof(data),
   &priority)) > 0){
        Printf("Received %d byte message >%s< with %d
   priority\n", n, data, priority);
     }
```

Messages are taken off the queue in order of their priority, which is returned by `mq_receive()`, into the variable passed to it. The return value of the function is the number of bytes in the message. In normal operation, this function would be called in a 'while' loop and the queue length would be checked at each iteration of the loop. The checking is done with the `mq_getattr()` function, called with the queue descriptor, and the atr structure, defined above:

```
    if(mq_getattr(qd, &atr)  == 0){
        if(atr.mq_curmsgs == 0){
           printf("No more messages\n");
            mq_close(qd);
        }
    }
```

The following code extract puts this all together:

```
        while((rval = mq_receive(qd, (char *)data,
    sizeof(data), &priority)) > 0){
            printf("Client received: >%s< priority %d\n",
    data, priority);
            memset(data, '\0', sizeof(data));
            if(mq_getattr(qd, &atr)  == 0){
                if(atr.mq_curmsgs == 0){
                    printf("No more messages\n");
                    mq_close(qd);
                    break;
                }
            }
        }
```

We now have all the information we need to write a test program that exercises all of these queuing functions. Instead of attempting to re-create MQ Series from scratch (which we will leave for the 'Advanced Queues' article), this program merely does the following:

- Create a queue, whose descriptor is 'qd'.
- Launch a child process, `child()` which asks to be notified of the arrival of a message
- Enqueue 4 messages, in ascending order of priority.
- The child pulls the messages off the queue, in the order that they arrived, i.e, in order of priority. It then quits.
- Launch another child process, `client()`, which merely performs a blocking read of the queue.
- Enqueue 4 more messages, in descending order of priority
- The child, again, pulls the messages off, in order of priority, which means the reverse of the order of their arrival. It does not quit.

Listing 1. *Server and Client Code*

```
#include <mqueue.h>
#include <sys/stream.h>
#include <sys/ddi.h>

void interrupt(int);        /* interrupt handler */

struct mq_attr atr;

char *msg1 = "Mary had a little lamb\n";
char *msg2 = "She also had a duck\n";
char *msg3 = "She put them on the mantelpiece\n";
char *msg4 = "To see if they would fall\n";

char *msg5 = "Mary had a little lamb\n";
char *msg6 = "full of fun and frolicks\n";
char *msg7 = "She threw it up into the air\n";
char *msg8 = "And caught it by its tail\n";

mqd_t qd;

main()                         /* main */

{

char data[255];
unsigned int priority;
int rval;
pid_t pid;
struct mq_attr xatr;

    signal(SIGURG, interrupt);

    atr.mq_maxmsg = 100;
    atr.mq_msgsize = 255;

    if((qd = mq_open("/zq", O_RDWR|O_CREAT, 0755, &atr))
   == (mqd_t)-1){
        perror("mq_open");
    }
    pid = child(qd);        /* this asks to get notified
   */

    sleep(1);        /* give the child time to stabilise
   */

    /* queue ordering is by priority, not time of
   arrival */
    if(mq_send(qd, msg4, strlen(msg4), 5) == -1){
        perror("mq_send");
    }
    if(mq_send(qd, msg3, strlen(msg3), 6) == -1){
        perror("mq_send");
    }
    if(mq_send(qd, msg2, strlen(msg2), 7) == -1){
        perror("mq_send");
    }
    if(mq_send(qd, msg1, strlen(msg1), 8) == -1){
        perror("mq_send");
    }

    sleep(1);        /* give the child time to exit */
    pid = client();  /* blocking, but no notification */

    /* these must arrive after the queue empties, or the
   child won't exit */
    if(mq_send(qd, msg5, strlen(msg5), 4) == -1){
        perror("mq_send");
    }
    if(mq_send(qd, msg6, strlen(msg6), 3) == -1){
        perror("mq_send");
    }
    if(mq_send(qd, msg7, strlen(msg7), 2) == -1){
        perror("mq_send");
    }
    if(mq_send(qd, msg8, strlen(msg8), 1) == -1){
        perror("mq_send");
    }

}                           /* main */

/********************************************************
Simple blocking read loop, which checks the queue length
   at each
 pass, and exits when it's empty.
 *******************************************************/

client()                    /* client */

{

pid_t pid;
char data[255];
unsigned int priority;
int rval;

    switch((pid = fork())){
        case -1:
```

```
        break;
        case 0:
            printf("Client collecting messages from
    queue...\n");
            /* this will block until the first msg
    arrives */
            while((rval = mq_receive(qd, (char *)data,
    sizeof(data), &priority)) > 0){
                printf("Client received: >%s< priority
    %d\n", data, priority);
                memset(data, '\0', sizeof(data));
                if(mq_getattr(qd, &atr)  == 0){
                    if(atr.mq_curmsgs == 0){
                        printf("No more messages\n");
                        mq_close(qd);
                        break;
                    }
                } else {
                    perror("mq_getattr");
                    break;
                }
            }
            printf("Done\n");
            mq_unlink("/zq");
            exit(0);
        break;
        default:
            return(pid);
        break;
    }

}                          /* client */

/********************************************************
The child asks to be notified of the arrival of a
   message, by
means of SIGURG, for which we've defined a handler. The
   child
then calls pause(), and waits for an interrupt. Inside
   the
interrupt handler, it performs blocking reads on the
   queue,
checking its length each time. When the queue is empty,
   it
returns, and calls mq_notify again, to turn off
   notification,
*  and permit the client routine to access the queue.
********************************************************/
child(qqd)                      /* child */

mqd_t qqd;

{

struct sigevent ev;
pid_t pid;

    switch((pid = fork())){
        case -1:
        break;
        case 0:
            printf("Child collecting messages from
    queue...\n");
            ev.sigev_notify = SIGEV_SIGNAL;
            ev.sigev_signo = SIGURG;

            if(mq_notify(qqd, &ev) < 0){
                perror("mq_notify");
            }
            pause();
            if(mq_notify(qqd, NULL) < 0){
                perror("mq_notify");
            }
            exit(0);
        break;
        default:
            return(pid);
        break;
    }

}                          /* child */

/********************************************************
Interrupt handler
We're only interested in SIGURG, for which we've been
   waiting
in pause().We perform our dequeuing function in this
   handler,
to save ourselves a function call, so it is important
   that the
queue variables be visible globally.
The mq_receive() loop performs reads the queue, checking
   its
length each time. When the queue is empty, we return.
 ********************************************************/
```

```
void
interrupt(what)          /* interrupt */

int what;

{

char data[255];
unsigned int priority;
int rval;

    printf("Received signal %d...\n", what);

    switch(what){
        case SIGURG:
            while((rval = mq_receive(qd, (char *)data,
   sizeof(data), &priority)) > 0){
                printf("Child received: >%s< priority
   %d\n", data, priority);
                memset(data, '\0', sizeof(data));
                if(mq_getattr(qd, &atr)  == 0){
                    if(atr.mq_curmsgs == 0){
                        printf("No more
   messages\n");
                        break;
                    }
                } else {
                    perror("mq_getattr");
                    break;
                }
            }
        break;
    }


}                             /* interrupt */
```

The notification mechanism uses a software interrupt defined by means of the sigevent structure. To do this, we first create the variable:

```
struct sigevent ev;
```

The interesting parts of this structure (defined fully in siginfo.h) are

```
struct sigevent {
      int sigev_notify;
      int  sigev_signo;
}
```

where sigev_notify has the values

```
SIGEV_NONE
SIGEV_SIGNAL
SIGEV_THREAD
```

We will choose `SIGEV _ SIGNAL`, since we want to catch an interrupt, with the arrival of each message on our queue. Later, if we need to turn off notification, we can do it by passing in `SIGEV _ NONE`.

Since sigev_signo lets us choose which signal can be sent to us, we'll choose something safe, that isn't used by other processes. SIGURG is normally sent out when an urgent condition exists on a socket or other I/O device and, in that capacity, is of no interest to us. Therefore, we will use SIGURG, and register it, together with our interrupt handler, in `main()`:

```
    signal(SIGURG, interrupt);
```

Then, in our `child()` function, when our child process is running, we define the kind of event we need, and the signal number that we're expecting, as follows:

```
            ev.sigev_notify = SIGEV_SIGNAL;
            ev.sigev_signo = SIGURG;
```

Immediately after these lines, we call `pause()`, which puts the process into a catatonic state, waiting for the arrival of an interrupt.

In reality, the server and client code would probably be in separate files, and run in unrelated processes. Since this is merely an exercise, all of the code is in one file, as follows.

MARK SITKOWSKI

Mark Sitkowski C.Eng, M.I.E.E Consultant to Forticom Security

How Secure can Secure Shell (SSH) be?

(OpenSSH VPN tunnelling)

This article is the third part of the series on OpenSSH and configurations and includes tricks which make using the protocol more secure. This article concentrates on Virtual Private Networks supported by OpenSSH.

What you will learn…

- How to configure VPN using OpenSSH.
- Good basics to make something new and secure on your own.

What you should know…

- Unix/Linux commands and SHELL environments.
- The basics of TCP/IP, routing, and VPN issues.
- Basic configuration of SSH (1st and 2nd parts of the article series)
- Understanding of security necessities.

VPN as Virtual Private Network is a set of protocols and apps to enable a (virtual) tunnel inside the network. In this case, the network means layer 2 and layer 3 of the OSI (Open System Interconnection) model but we are focusing on layer 3, VPN tunnel. Admittedly, the OpenSSH supports layer 2 tunnelling, but for ease of use and understanding, this article will focus on layer 3 tunnelling.

Please look at the depicted Figure 1. There is a scheme of the small network configuration where our OpenSSH tunnel is through the Internet. It means that two separated private networks are connected directly via Internet and packets are routed to the appropriate network to the other side. The goal is to ensure secure traffic between 10.0.0.0/24 and 172.16.0.0/24 networks. VPNs can provide protection in unsecure networks as well.

OpenSSH is very configurable and we can use it independently of existing SSH configuration in order not to disturb a terminal access client/server model (further explanation is in article 1 of the series – issue 11/2013 of BSD Magazine). Whole traffic between these networks is as secure as OpenSSH protocol is secure. Therefore, encryption is enabled and no one can easily understand what we send through the Internet. Be informed,the OpenSSH team quotation from man ssh advises: *Since an SSH-based setup entails a fair amount of overhead, it may be more suited to temporary setups, such as for wireless VPNs. More permanent VPNs are better provided by tools such as ipsecctl(8) and isakmpd(8).* So, we can use it for small traffic but a large amount of bandwidth.

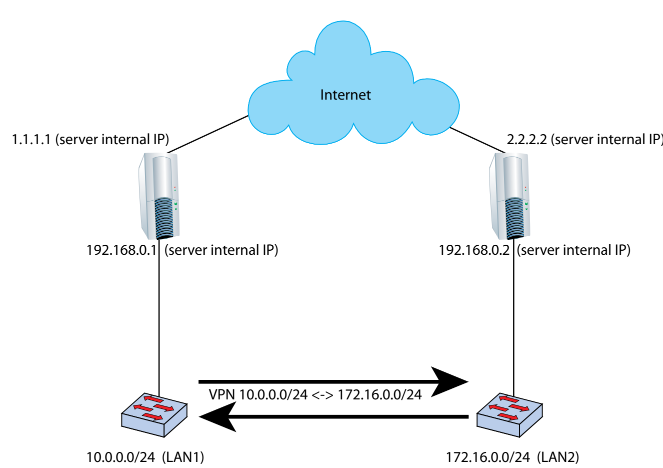


Figure 1. *Network schema of VPN tunnelling*

IP SETTINGS CONFIGURATION

First, we have to create virtual interfaces for temporary use and potentially for future use. Both interfaces should be made up on the server and client side. To do it for temporary use (until first system reboot or `/etc/netstart` command release) type the following commands:

```
server# ifconfig tun0 create
server# ifconfig tun0 192.168.0.1 192.168.0.2 netmask
   255.255.255.252
```

The results should be similar to Listing 1.

Secondly, we should be sure the forwarding is enabled on both sides. To check it, run the command shown below.

```
server# sysctl | grep ip.forwarding
```

Output (required):

```
net.inet.ip.forwarding=1
```

If the result is equal to 0, then run the following command.

```
server# sysctl net.inet.ip.forwarding=1
```

Listing 1. *Output of ifconfig tun0 pseudo-device interface on the server side*

```
server# ifconfig tun0
tun0: flags=11<UP,POINTOPOINT> mtu 1500
        priority: 0
        groups: tun
        status: down
        inet 192.168.0.1 --> 192.168.0.2 netmask
   0xfffffffc
```

Listing 2. *Output of ifconfig tun0 pseudo-device interface on the client side*

```
server# ifconfig tun0
tun0: flags=11<UP,POINTOPOINT> mtu 1500
        priority: 0
        groups: tun
        status: down
        inet 192.168.0.2 --> 192.168.0.1 netmask
   0xfffffffc
```

Output:

```
net.inet.ip.forwarding: 0 -> 1
```

To set it permanently add the line `net.inet.ip.forwarding=1` into the `/etc/sysctl.conf` file.

For the client side, check whether forwarding is enabled and then create the pseudo-device interface tun0. The command sequence is as follows; results of the commands are shown in Listing 2:

```
client# ifconfig tun0 create
client# ifconfig tun0 192.168.0.2 192.168.0.1 netmask
   255.255.255.252
```

Thirdly, for future use of pseudo-device at start up after reboot or similar, create the following file at OpenBSD or modify specified file at FreeBSD.

OpenBSD (on server and client side)

```
server# echo "192.168.0.1 192.168.0.2 netmask
   255.255.255.252" > /etc/hostname.tun0
client# echo "192.168.0.2 192.168.0.1 netmask
   255.255.255.252" > /etc/hostname.tun0
```

FreeBSD (on server and client side)

```
server# echo 'ifconfig_tun0="inet 192.168.0.1 192.168.0.2
   netmask 255.255.255.252"' >> /etc/rc.conf
client# echo 'ifconfig_tun0="inet 192.168.0.1 192.168.0.2
   netmask 255.255.255.252"' >> /etc/rc.conf
```

Last but not least, set up the appropriate routing table for both server and client. Let's look at Figure 1 again to understand better what we should do and along with the packets' destination. For temporary use commands are as follows:

OpenBSD (on server and client side)

```
server# route add 172.16.0.0/24 192.168.0.2
client# route add 10.0.0.0/24 192.168.0.1
```

FreeBSD (on server and client side)

```
server# route add -net 172.16.0.0/24 192.168.0.2
client# route add -net 10.0.0.0/24 192.168.0.1
```

To set the permanent routing entries (static routes) after reboot etc., modify your configuration files with the following commands:

OpenBSD (on server and client side)

```
server# echo "!route add 172.16.0.0/24 192.168.0.2 > /dev/
   null 2>&1" >> /etc/hostname.tun0
client# echo "!route add 10.0.0.0/24 192.168.0.1 > /dev/
   null 2>&1" >> /etc/hostname.tun0
```

FreeBSD (on server and client side)

```
server# echo 'static_routes="vpn1"' >> /etc/rc.conf
server# echo 'route_vpn1="-net 172.16.0.0/24 192.168.0.2"'
   >> /etc/rc.conf
client# echo 'static_routes="vpn1"' >> /etc/rc.conf
client# echo 'route_vpn1="-net 10.0.0.0/24 192.168.0.1"'
   >> /etc/rc.conf
```

This is the end of the discussion on IP settings for VPN tunnelling, so let's begin to prepare OpenSSH server and then SSH client to negotiate and start tunnelling.

Openssh: Server And Client Configuration

This section of the article focuses on configuration of the SSH server and client, which is the same for both OpenBSD and FreeBSD operating systems. Let's assume that we use OpenSSH as a server for a terminal use, a file transfer or even another VPN tunnelling connection as well as an all-in-one.

It's good to know that we can use a separate sshd process started with a specific defined configuration file and use a different server port. For example, we use standard SSH port 22 for terminal connections and we can use non-standard 2468 port for VPN connections. The configuration file mentioned above can be different as well, so we can forget about any existing SSH connections, configuration etc. and start to use it only for VPN tunnelling.

Server

The first step is to copy the existing configuration file sshd_config to the new file:

```
server# cp /etc/sshd_conig /etc/sshd_config_vpn
```

After that we need to change some options and values, so edit the new file `sshd_config_vpn` and add/change the following lines.

You should be familiar with these options, described in the 1st in the series (issue 11/2013 of BSD Magazine). There are two new options PermitTunnel and AllowTcpForwarding responsible for enabling tunnelling and forwarding packets relatively.

```
PermitTunnel point-to-point
Port 2468
ListenAddress 1.1.1.1
AllowUsers root
PermitRootLogin yes
AuthenticationMethods publickey
AllowTcpForwarding yes
```

On the server side we generate the new private/public key, which we will use to start securing SSH connections. That is the same step described in the 1st article as well. The command generating these keys is as follows (Please leave the passphrase empty to prevent continuously being asked for that during every VPN connection):

```
server# ssh-keygen -b 4096
```

As described in the 1st article, copy a public key file to authorized_keys file and private file into the client file system.

Client

The next step is to copy existing configuration file ssh_config to the new file:

```
server# cp /etc/ssh_conig /etc/ssh_config_vpn
```

We need to change a couple of options and values as well. Edit the file ssh_config_vpn and add/modify the following lines.

```
Port 2468
Protocol 2
Tunnel point-to-point
PasswordAuthentication no
AddressFamily inet
IdentityFile /my_own_path_to_ssh/private_key
TunnelDevice 0:0
```

Some explanation is needed for the TunnelDevice option. This option is asking for what pseudo-device interface number should be used for both sides. 0:0 means for tun0.

After that we are ready to run our OpenSSH VPN tunnel. Let's run the following command from the client.

```
client# ssh -v -F /etc/ssh/ssh_config_vpn -l root 1.1.1.1 true
```

To troubleshoot connection problems it is good to set -v option in order to output more debug data during creation of the VPN connection. A successful setting of VPN tunnelling is shown on Listing 3.

If everything works great, we can do some hardening: running VPN at start up and prevent to login as any user, especially root to terminal on the other side, just allow only to create VPN.

To run the VPN tunnel after reboot, etc., we should do as follows (commands for OpenBSD and FreeBSD):

OpenBSD (on the client side)

```
client# echo "/usr/bin/ssh -F /etc/ssh/ssh_config_vpn -l
   root 1.1.1.1 true" >> /etc/rc.local
```

FreeBSD (on the client side)

```
client# echo "#!/bin/sh" >> /usr/local/etc/rc.d/vpn.sh
client# echo ". /etc/rc.subr" >> /usr/local/etc/rc.d/vpn.sh
client# echo "rcvar=sshvpn_enable"
client# echo 'command="/usr/bin/ssh -F /etc/ssh/ssh_config_
   vpn -l root 1.1.1.1 true"' >> /usr/local/etc/rc.d/vpn.sh
client# chmod 550 /usr/local/etc/rc.d/vpn.sh
client# echo 'sshvpn_enable="YES"' >> /etc/rc.conf
```

The last thing is to use SSH connection for VPN tunnelling only. To do that we have to change the following line on the server side in the file sshd_config.

PermitRootLogin forced-commands-only

And on the client side add/modify the following line at the file ssh_config_vpn.

```
tunnel="1",command="sh /etc/netstart tun0" ssh-rsa
```

CONCLUSIONS

Virtual Private Networks are good solutions to provide secure and low cost internal traffic between branches. OpenSSH is one of the many such worthwhile methods for using VPN tunnels but not the best. You can use it for small networks with low traffic between sites. You can use it as a secure gateway to enable new traces as well for security purposes only. OpenSSH is very flexible so it's good to concatenate SSH terminal connections with VPN tunnelling to improve your security access into the system. You can try to make up the fake traffic as circumstances for threats and thus decrease your system's vulnerabilities.

This part is the last about strictly securing OpenSSH. The last one will explain why OpenSSH used for SFTP (SSH File Transfer Protocol) is better than FTP or even FTPS.

In the next series you will find out more about: SFTP – known as SSH File Transfer Protocol to opposite of a standard FTP.

Listing 3. *Successful setting of VPN connection, data from the server side*

```
tun0: flags=51<UP,POINTOPOINT,RUNNING> mtu 1500
        priority: 0
        groups: tun
        status: active
        inet 192.168.0.1 --> 192.168.0.2 netmask
   0xfffffffc
```

References (order of relevance):

- man sshd_config (server side configuration file)
- man ssh_config (client side configuration file)
- man sshd (server side binary file)
- man ssh (client side binary file)
- *www.openssh.org*

ARKADIUSZ MAJEWSKI, BENG

Arkadiusz Majewski comes from Poland. He has 15 years of experience with ICT technologies, including 15 years of IT networks, 10 years of BSD systems and MS Windows Server solutions. He also has 5 years of experience with programming languages and Telco solutions. He's interested in security on all business and ICT levels. In his free time, he reads ICT books and deepens his knowledge about science (math, physics, chemistry). His hobby is cycling and motorization. He's a graduate of Warsaw Information Technology under the auspices of the Polish Academy of Sciences. Feel free to contact the author via e-mail bsd.magazine@iptrace.pl.

Unix Interprocess Communication Using Shared Memory

A shared memory segment is a section of RAM, whose address is known to more than one process. The processes to which this address is known, have either read only, or read/write permission to the memory segment, whose access rights are set in the manner used by chmod.

Most machines dedicated to manipulation of large databases are not short of RAM, and figures of 3 to 5 GB are fairly common. Where two processes coexist on the one machine, communication of data through the mechanism of shared memory becomes an attractive proposition.

Among the advantages of a shared memory system are:

- Memory-to-memory data transfers are inherently fast, and there are never any connection problems, as can occasionally occur with TCP/IP.
- The total amount of memory used by a TCP/IP client server system, in the worst case, is double the amount necessary to store the data. First, the client has to extract the data, and store it in local data structures, like arrays of structures, or linked lists and, then, the server has to allocate the same amount of memory, to receive the same data. Memory is returned only when the client terminates.

The drawbacks include:

- The amount of free RAM must always be adequate to cater to the maximum which may be required.
- If a process terminates unexpectedly without first deleting its shared memory segment, that segment remains unusable. If the segment is of significant size, this could have an adverse effect on the performance of the machine.
- The parent/child interaction, at the beginning of the operation is slightly more complicated. The child needs to communicate the address of the shared memory segment, which it has allocated for the data it is about to send back to the parent. In order for this to be possible, the parent must, first, establish a small piece of shared memory, where the child can place this address.
- The timing of connections and disconnections is not event-driven.

Shared Memory Commands

A shared memory segment is requested with the shmget() system call, which has the synopsis:

```
Int shmget(key_t key, size_t size, int shmflg);
```

The return value is the shared memory identifier, an integer value, which is used in subsequent manipulations.

On some versions of Unix, the 'key' parameter can be synthesized by calling a special function, but for most purposes and certainly for ours, the symbolic value IPC_PRIVATE, which is #defined as zero, will be exclusively used.

The variable 'size' is merely the memory segment size in bytes while the 'shmflag parameter is the logical OR of one or more of the following:

`IPC_CREAT` – create segment if key doesn't exist
`IPC_EXCL` – fail if key already exists
`IPC_NOWAIT` – flag error if we must wait for the segment
`SHM_R` – make segment readable
`SHM_W` – make segment writeable
`SHM_RND` – attach on page boundary
`SHM_RDONLY` – attach as read-only. If this is omitted, the default is read/write.
`SHM_SHARE_MMU` – share virtual memory among processes which share this segment. This may be useful, if there is a danger of one or more of such processes being swapped out.
`SHM_PAGEABLE` – As above, but the memory may be dynamically resized within the size allocated.

Typically, we would make the call as follows:

```
#include <shm.h>
int shmid;
size_t size = 10000000;

if((shmid = shmget(IPC_PRIVATE, size, IPC_CREAT | SHM_
   PAGEABLE |

   SHM_R | SHM_W)) <= 0){
      perror("Error obtaining shared memory");

  }
```

Having acquired our shared memory, we now have to attach it to the data segment of our process. This is achieved by using the `shmat()` system call.

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

The return value is a pointer to the start address of the attached memory segment. It is declared (void *) for the same reason as that of `malloc()`. It is the responsibility of the user to cast this to the datatype for which the memory will be used.

The 'shmid' parameter is that returned from the shmget call, above, while shmaddr has the following common options:

- shmaddr = 0 the segment is attached to the first available suitably aligned address.
- shmaddr != 0 AND shmflag is either SHM_SHARE_MMU (which means the kernel will share its unpageable memory resources) or SHM_PAGEABLE (memory is pageable), the segment is attached to the first suitably aligned address at shmaddr. This is the most commonly used value, and one we shall use.

The shmflag argument can have most of the values passed to `shmget()`:

```
SHM_R | SHM_W | SHM_RDONLY | SHM_RND | SHM_SHARE_MMU |
   SHM_PAGEABLE
```

We will use `shmat()` as follows:

Listing 1. *A structure of type struct shmid_ds, which may be used to obtain information about the memory segment*

```
struct shmid_ds {
        struct ipc_perm shm_perm;           /* permissions struct */
        size_t          shm_segsz;          /* size of segment I(bytes) */
        struct anon_map *shm_amp;           /* segment anon_map pointer */
        ushort_t        shm_lkcnt;          /* number of times it is being locked */
        pid_t           shm_lpid;           /* pid of last shmop */
        pid_t           shm_cpid;           /* pid of creator */
        shmatt_t        shm_nattch;         /* used only for shminfo */
        ulong_t         shm_cnattch;        /* used only for shminfo */
        time_t          shm_atime;          /* last shmat time */
        time_t          shm_dtime;          /* last shmdt time */
        time_t          shm_ctime;          /* last change time */
};
```

Listing 2. *The code for child processes*

```
/* the shmid of the token memory being passed to the
   child */
Int shmid_s;

/* an array for storing pointers to all the tokens,
   passed to all child processes */
Unsigned char chptr[NCHILDREN];


Server(char *cursor_SQL_string, int which_cursor)

{

 /*
     * token memory, so for child to write its ID and
   shmid
     */
    token = sizeof(unsigned char) * 200;

    /*
     * shmid_s is global, so it can be viewed by the
   child process,
     * and attached.
     */
    if((shmid_s = shmget((key_t)IPC_PRIVATE, token, IPC_
   CREAT | 0666)) <= 0){
        perror("Server: Error obtaining shared memory");
        return(-1);
    }

    /*
     * shmat returns a pointer to the segment defined by
   shmid
     */
    if((chptr[which] = (unsigned char *)shmat(shmid_s,
   0, SHM_RND)) == (unsigne
d char *)-1){
        perror("Server: Error attaching to shared
   memory");
        return(-1);
    }

/* the next line cleans the memory we're going to use */

 memset((char *)chptr[which], '\0', token);

     /* launch child process */

 switch((pid = fork())){
     case -1:
          perror("Fork");
     break;
     case 0:                    /* in child process
*/
           /* connect to database, prepare cursor
from cursor string,
            * declare it, and open it
           */

           /* attach the child to the token memory */

           if((chptr[which_cursor] = (unsigned char
*)shmat(shmid_s, 0, SHM_PAGEABLE)) == (unsigned char
*)-1){
              perror("Client: Error attaching to
incoming shared memory");
              exit(-1);
           }
           switch(which_cursor){
                case 1:
                     /* run SQL query to determine
number of rows to be returned */

                     /* allocate shared memory to
hold all the data */

                     if((shmid = shmget(IPC_PRIVATE,
size,  IPC_CREAT | 0666)) <= 0){
                          printf("Memory allocation
failed\n");
                          quit(-1);
                     }
                       /*
                        *  now get a pointer to the
actual memory,
                        *  cast to the data type of
the structure we expect to receive
                        */
                       if((mpt = (struct xyz *)
shmat(shmid, 0, SHM_RND)) == (struct xyzg *)-1){
                           perror("Client: Error
attaching to shared memory");
                           quit(-1);
                       }

                         /* code to fetch cursor into
the mpt[] array of structures */
```

```
                    /* when all rows have been
retrieved, write results to token */
                    sprintf((char *)
chptr[which], "%d %d %d", which, rcnt, shmid);
                break;
                case 2:
                    /*  code for second cursor,
which has
                     *  different cursor string
and data structures
                     */
                break;
                case 3:       /* etc …. */
                 break;
                 default:
                     printf("Unknown cursor\n");
                  break;
            }
        /* code to close the cursor */
        break;
        default:      /* in the parent process */
            children++
         break;
    }

}
```

Listing 3. *The presence of all three signifies that the cursor in the child has run, and that data is available*

```
monitor()                      /* monitor */

{

int one, two, three;                 /* dummy variables
   for testing token */
int flag = 0;                            /* termination
   flag */
int done[NCHILDREN];           /* log of completed
   children */

    printf("Server waiting for clients to connect shm
   segments...\n");
    th = 0;
    memset((char *)done, '\0', sizeof(done));

    while(1){
        for(i = 1; i <= NCHILDREN; i++){
            if(sscanf((char *)chptr[i], "%d %d %d",
&one, &two, &three) == 3){
                if(one == 0 || two == 0 || three == 0)
continue;
                if(done[i] == 99) continue;
                printf("Child %d returned\n", i);
                children--;
                done[i] = 99;            /* mark this
child as having completed */

                /* let a thread deal with this, while we
continue to look */
                if((pthread_create(&thr[th], NULL,
xserve, (void *)chptr[i])) !
= 0){
                    printf("Failed to create thr[%d]\
n",th);
                }
                /* we don't want to wait for the thread
*/
                if(pthread_detach(thr[th]) != 0){
                    printf("Failed to start thr[%d]\
n",th);
                }
                th++;
                printf("Server %d: Threads running: %d
Children: %d\n",

getpid(), th, children);
            }
            if(children == 0){        /* all our cursors
have run, so process the data */
                flag = 1;
                break;
            }
        }
        if(flag == 1) break;
    }
}
```

```
unsigned char *mptr;

if((mptr = (unsigned char *)shmat(shmid, 0, SHM_RND)) ==
   (unsigned char *)-1
){
        perror(" Error attaching to shared memory\n");
 }
```

Unlike malloc, which returns NULL on failure, shmat returns –1, which results in the need for the clumsy cast to (unsigned char *), above.

Each attached memory segment has associated with it, a structure of type struct shmid_ds, which may be used to obtain information about the segment: Listing 1.

The `shmctl()` system call, is designed to load the contents of this structure into a local structure of the above type:

```
if(shmctl(shmid2, IPC_STAT, &buf) < 0){
     printf("Unable to get shm status\n");
 }
```

The variable IPC_STAT signifies that this is a query. The variable IPC_SET allows the setting of the members of the ipc_perm structure, and changing the following permissions:

```
shm_perm.uid
shm_perm.gid
shm_perm.mode
```

Still considering our hypothetical database access program, described at the beginning of this chapter, the sequence of events, for creating a shared memory client-server system, would be:

- Parent process allocates a 100-byte shared memory segment, large enough to hold a token, with the child's ID, the number of bytes, or data structures being returned and the shared memory ID allocated and returned by the child
- Parent forks child processes, each of which is passed the shared memory ID of the 100-byte token memory segment.
- Child process accesses the database, and queries the number of rows which will be returned by the cursor, which it intends to run.
- Child process allocates shared memory, large enough to hold the data, then retrieves the data from the database, and loads it into the memory segment.

Listing 4. *To access our data*

```
Void *
Xserve(unsigned char shm)                           /
   xserve */

{

int cur_id;
int size;
int shmid_c;

  /* extract token data */
     sscanf((char *)shm, "%d %d %d", &cur_id, &size,
    &shmid_c);
     printf("Thread %d cursor %d shmid %d ..\n",
    pthread_self(), cur_id, shmid_c);

     /*
     * shmat returns a pointer to the segment defined by
    shmid_c
     */
    if((data[cur_id] = (unsigned char *)shmat(shmid_c,
    0, SHM_RND)) == (unsigne
d char *)-1){
          perror("Server: Error attaching to shared
     memory");
          return((void *)-1);
      }

      /*
       * Cast pointers to correct data types,
       * and set no. of records
       */
      switch(cur_id){
          case 1:
              tpt = (struct xyz *)data[cur_id];
              lpt = size;
          break;
          case 2:
              /* same for next cursor */
          break;
          case 3:   /* etc… */
           break;
        }
}
     /* xserve */
```

- Child process places its identifier, the number of rows being returned and the shared memory ID of the retrieved data in the 100-byte token memory segment.
- Parent reads the child's identifier, the number of rows being returned and the shared memory ID. It then attaches to the shared memory segment and accesses the data.

Server

This code would probably reside in the routine which launched child processes, and require the following global declarations: Listing 2. The above routine would be called once for every cursor and after the last call, each element of the array `chptr[]` would contain a pointer to the shared memory tokens, passed to all the children. We would then call a monitor routine, which would scan the elements of the array, looking for a child identifier, a row count and a shmid. The presence of all three signifies that the cursor in the child has run, and that data is available (Listing 3).

We send a thread to perform the housekeeping on the data that has just arrived, so that we can continue to search uninterrupted for returned children.

In the function `xserve()`, we attach to the memory segment, defined by the shmid, returned in the token. We store the pointer, returned by `shmat()`, in a global array of such pointers, which we will use in the subsequent data manipulation routines, to access our data (Listing 4).

MARK SITKOWSKI

Mark Sitkowski C.Eng, M.I.E.E Consultant to Forticom Security.

Sniffing and Recovering Network Information Using Wireshark

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, as well as education. Wireshark is cross-platform, using the GTK+ widget toolkit to implement its user interface and pcap to capture packets, it runs on various Unix-like operating systems including Linux, OS X, BSD, Solaris, and on Microsoft Windows.

You can download Wireshark for Windows or Mac OS X from the official website (*http://www.wireshark.org/download.html*). Most Linux systems come with a pre-installed Wireshark tool; however, in the case that Wireshark is not installed, you can just follow the documentation below and run the proper command for each operating system to get it running: Building and Installing Wireshark (*http://www.wireshark.org/docs/wsug_html_chunked/ChapterBuildInstall.html*). Wireshark needs to be run as the root user in your system and will give you a security message that you are running it as root, so proceed with proper caution.

Capture Interfaces

We can get an overview of the available local interfaces by navigating on the Capture menu tab and then clicking the Interfaces option as shown in Figure 1. By clicking the Option button, Wireshark pops up the "Capture Options" dialog box. The table shows the settings for all available interfaces including a lot of information for each one and some checkboxes like:

- Capture on all interfaces – As Wireshark can capture on multiple interfaces, it is possible to choose to capture on all available interfaces.
- Capture all packets in promiscuous mode – This checkbox allows you to specify that Wireshark should put all interfaces in promiscuous mode when capturing.

By clicking the Start button, we will see a lot of packets start appearing in real time. Wireshark captures each packet sent from (Source) or to (Destination) our system.

User Interface

Before proceeding to analyze our traffic network we will explain the basic information we need to know about the packet list pane, the color rules, the packet details pane and the packet bytes pane.

Packet List pane

The packet list pane displays all the packets in the current capture file. Each line in the packet list corresponds to one packet in the capture file. If you select a line in this

pane, more details will be displayed on Packet Details and Packet Bytes panes.

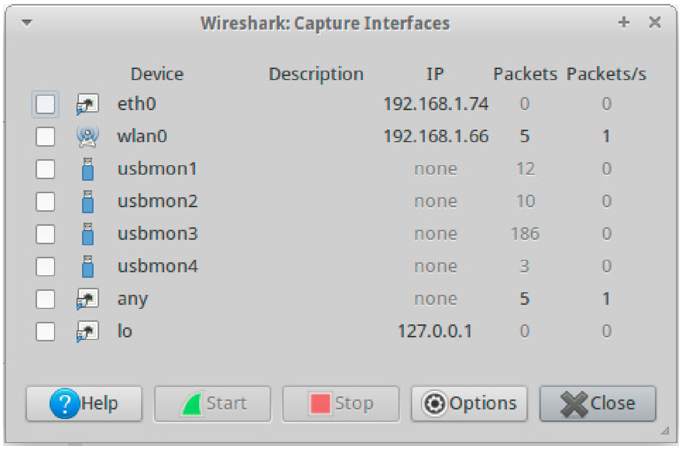


Figure 1. *Wireshark Interfaces*

The default columns will show:

- No. – The number of the packet in the capture file. This number won't change, even if a display filter is used.
- Time – The timestamp of the packet. The presentation format of this timestamp can be changed.
- Source – The address where this packet is coming from.
- Destination – The address where this packet is going to.
- Protocol – The protocol name in a short (perhaps abbreviated) version.
- Info – Additional information about the packet content.

Color Rules

A very useful mechanism available in Wireshark is packet colorization. There are two types of coloring rules in Wireshark; temporary ones that are only used until you quit the program, and permanent ones that will be saved to a preference file so that they are available on a next session. So let's focus on the most important name filters. Green Color refers to TCP packets but black identifies corrupted TCP packets. Light Blue refers to UDP packets and dark blue on DNS traffic. For more information or to edit/add our own color rules, we can navigate to View menu and click the *Coloring Rules*.

Packet Details Pane

The packet details pane shows the current packet (selected in the "Packet List" pane) in a more detailed form. This pane shows the protocols and protocol fields of the packet selected in the "Packet List" pane. The protocols and fields of the packet are displayed using a tree, which can be expanded and collapsed.

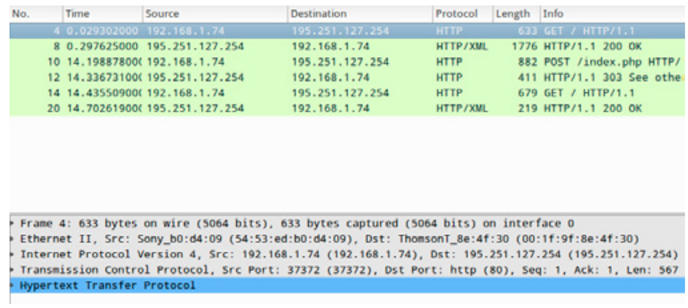


Figure 2. *List – Details Pane*

Packet Bytes Pane

The packet bytes pane shows the data of the current packet in a hexdump style. The left side shows the offset in the packet data, in the middle the packet data is shown in a hexadecimal representation and on the right the corresponding ASCII characters are displayed.

Start Capturing – Analyzing

In this part we will start capturing once more on our network, so click from Capture menu the Start option. Next we will attempt to log in to an account and analyze it into the Wireshark tool to see if we can find important information. As we can see there are a lot of packets that Wireshark appears. A valuable option here is the Filter mechanism which lets us quickly edit and apply display filters. Let's isolate the http packets by typing http string on filter tab. As we can see, the packet list pane shows only HTTP protocols. We need to locate the HTTP protocol and identify the response of the Host which attempted to log in. Looking at the highlighted results, we can determine at the info tab that there are packages which contain the GET method. Let's focus on this information and explain it.

Note

GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. At the packet list pane, click the Hypertext Transfer Protocol. As we can see, the GET method appears and also a lot of important information such as the request version of the Server, the Host and the User-Agent which contains the browser version and the OS that the user used to login. Next we want to examine the full conversation between the client and the server by accessing the Follow TCP Stream option (right click on the packet and then choose Follow TCP Stream).

A pop-up window will appear which will contain the entire conversation on stream content. The red words indicate the request and the blue, the response of the Host. Also as we can notice, choosing the Follow TCP Stream option Wireshark automatically added the property filter in Filter area.

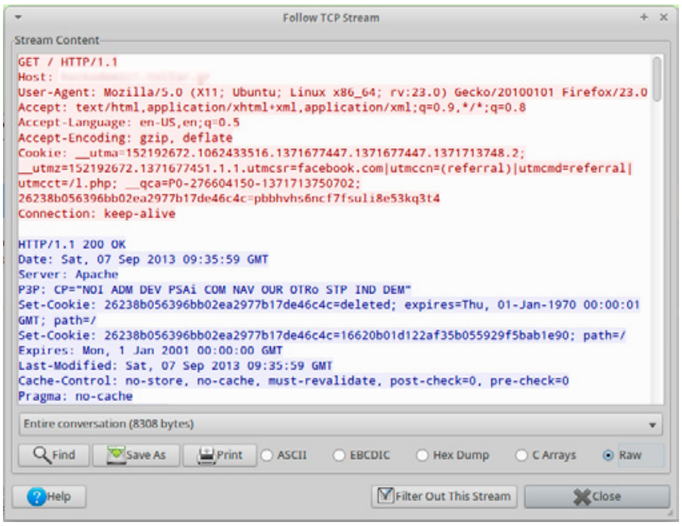


Figure 3. *TCP Stream Window*

By reviewing the highlighted code closely on Figure 3, we can see that the index.php action has two inputs, the username and the password. We can identify on Packet List pane a POST Request method from our machine to the server using HTTP protocol. Selecting once more the Hypertext Transfer Protocol tree, we can verify the request and the method which was used to login to the Host.

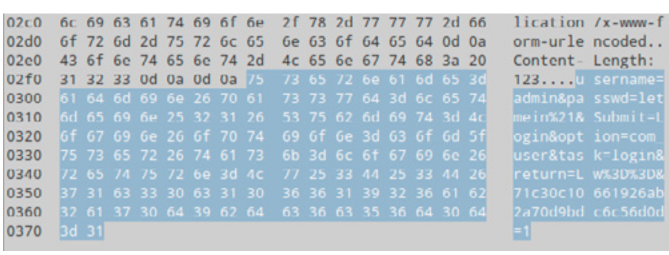


Figure 4. *Bytes Pane.*

Note

POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.

As we can notice on the packet details pane, there is also a new tree line named Line-based text data. By clicking once, we can see the POST request which contains the username and the password in clear text. Also checking the packet bytes pane we can draw the same information on Hex or Bit View.

Cracking – Analyzing W-Network

In this part of the article, we will explain how we can have access to our WLAN network, how to retrieve the wireless password and, finally, how we can use it to analyze the traffic packets into Wireshark.

First we will run the following command to get a list of our network interfaces:

```
wizard32@wizard32:~$ sudo airmon-ng
Interface Chipset       Driver
wlan0      Unknown    iwlwifi - [phy0]
```

As we can notice the only available interface is the wlan0 adapter. To capture network traffic without being associated with an access point, we need to set the wireless network adapter in monitor mode (Listing 1).

Next run the Wireshark tool once more and navigate to the Capture menu and click the Interfaces option. As we mentioned before, monitor mode enabled on mon0 so on wireshark pop-up window select the mon0 as capture interface and click start (Figure 5). After starting the capture,

Listing 1. *Setting wireless network adapter in monitor mode*

```
wizard32@wizard32:~$ sudo airmon-ng start wlan0

Found 4 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working
    after
a short period of time, you may want to kill (some of)
    them!

PID Name
1103   NetworkManager
1121   avahi-daemon
1125   avahi-daemon
1299   wpa_supplicant


Interface Chipset       Driver

wlan0      Unknown    iwlwifi - [phy0]
                (monitor mode enabled on mon0)
```

we locate multiple SSID access points. By typing HTTP or DNS on Filter menu, Wireshark doesn't return any result. Looking on the packet list pane, we can search our access point or by locating the BSSID (basic service set identification) or the SSID (service set identifier).

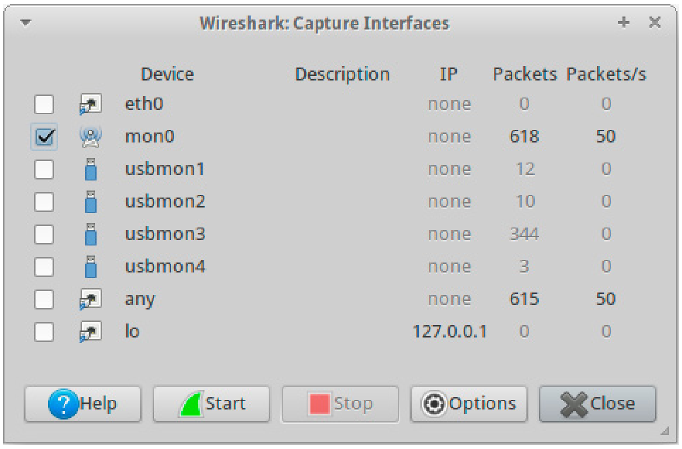


Figure 5. *Wireshark Interfaces*

- BSSID is the MAC address of the wireless access point (WAP) generated by combining the 24 bit Organization Unique Identifier and the manufacturer's assigned 24-bit identifier for the radio chipset in the WAP.
- SSID is the name of a wireless local area network (WLAN).

As we can notice, two new tree lines have been added on the packet details pane. Both of them specify the communication wireless protocol.

Another way to locate our access point is to use the airdump-ng tool.

```
wizard32@wizard32:~$ sudo airodump-ng mon0
BSSID              PWR  Beacons    #Data, #/s  CH  MB
   ENC  CIPHER AUTH ESSID
00:11:8F:8E:4E:32  -30        21         0    0   1  54
   WEP  WEP         wizard32
```

Listing 2. *Retrieving WEP network key*

```
wizard32@wizard32:~$ sudo aircrack-ng ~/Desktop/W-packets-01*.cap
Opening /home/wizard32/Desktop/W-packets-01.cap
Read 61960 packets.

   #  BSSID              ESSID                     Encryption

   1  00:11:8F:8E:4E:32  wizard32                  WEP (21124 IVs)

Choosing first network as target.

Opening /home/wizard32/Desktop/W-packets-01.cap
Attack will be restarted every 5000 captured ivs.
Starting PTW attack with 21124 ivs.

                    Aircrack-ng 1.1
                                                     00:00:02] Tested 7 keys (got 21124 IVs)

   KB    depth   byte(vote)
    0    0/  1   4B(29696) E4(28160) 40(27648) C2(27392) D6(26368) 21(26112) 62(25344) A8(25344) B3(25344) DB(25344)
   2C(25088) 38(25088) A9(25088) 47(24832) C4(24832) CB(24832) CE(24832) 19(24320) 44(24320)
   [...]
    4    0/  2   C4(29440) 12(28928) 78(28160) 87(27136) 60(26368) 84(26368) 93(25856) 00(25600) 4C(25600) BD(25344)
   C5(25344) 03(25088) 68(25088) 7B(25088) F4(25088) 02(24832) 1E(24832) 28(24832) 54(24832)
   [...]

                         KEY FOUND! [ 4B:AB:FE:1C:02 ]
   Decrypted correctly: 100%
```

To capture data into a file using the airodump-ng tool once more, we must specify some additional option to target a specific access point.

```
wizard32@wizard32:~$ sudo airodump-ng -c 1 -w ~/Desktop/W-
    packets --bssid 00:11:8F:8E:4E:32 mon0
```

Currently, we can use two different ways to retrieve the password from our network. The first one is to use a tool named aircrack-ng in association with the .pcap packets that we captured using the aiodump-ng tool or using the .pcap file from the Wireshark tool and performing a dictionary attack to a specific access point. Let's analyze them.

Method: aircrack-ng

To recover the WEP key aircrack only requires the collection of enough data. So, in the terminal we type the following command to retrieve our WEP network key: Listing 2. As we can see, aircrack decrypted and correctly found our WEP network key. Let's analyze how we can retrieve it using the dictionary attack method on .pcap Wireshark file (Listing 3) this time.

```
-w: Identifies our wordlist file
```

Note

Some of these tools (airmon-ng) might need to be installed, unless we are using a system which has airmon-ng already installed, such as BackTrack/Kali or BackBox.

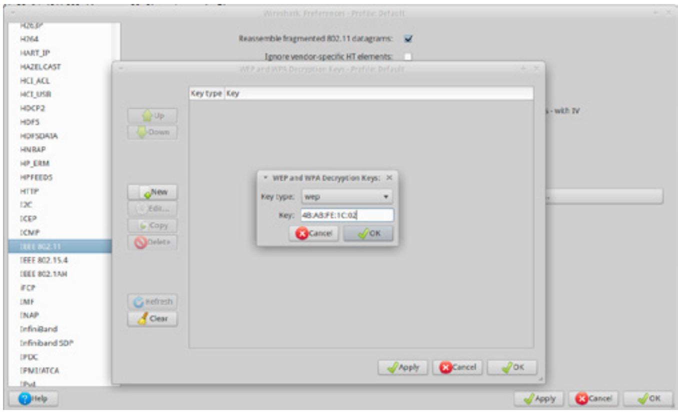


Figure 6. *Decryption Keys Pane*

In both cases, aircrack successfully recovered the WEP key. Now it's time to apply our WEP key into Wireshark tool to enable decryption to locate possible sensitive information. Navigate to Edit menu, then click on Preferences option and on Protocol tree line locate the IEEE 802.11 protocol. Next we mark the Enable decryption checkbox and then we click the Edit button to add our WEP key.

The Moment of Truth (TMT)

We are searching once more for possible http || dns protocols. By reviewing the highlighted code closely on figure 2 we can see multiple http requests to a specific host. To eliminate even more results we will create a new filter which will specify only those packages from the specific Host. So we locate the GET request and we apply the selected

Listing 3. *Retrieving the WEP network key using the dictionary attack method*

```
wizard32@wizard32:~$ sudo aircrack-ng -w ~/Desktop/mywordlist.txt -b 00:11:8F:8E:4E:32 ~/Desktop/W-capture.pcap
Opening /home/wizard32/Desktop/W-capture.pcap
Attack will be restarted every 5000 captured ivs.
Starting PTW attack with 21096 ivs.

                                                                     Aircrack-ng 1.1
                                                 [00:00:02] Tested 7 keys (got 21096 IVs)

   KB    depth   byte(vote)
    1    0/  1   AB(34816) 32(27904) C6(27648) B0(26624) 12(26112) 16(25600) 28(25600) B1(25600) CD(25344) F5(25344)
   60(25088) D0(25088) E1(25088) D4(24832) 20(24576) 10(24320) 82(24320) 21(24064) 4A(24064)
   [...]
    2    2/  3   FE(27648) 4A(26624) B9(25600) EB(25600) 0D(25344) 2A(25344) 3A(25344) 46(25088) 25(24832) 7B(24832)
   8E(24832) 9A(24832) AF(24832) 01(24576) C1(24576) 5E(24320) 78(24320) 8F(24320) BD(24320)
   [...]

                       KEY FOUND! [ 4B:AB:FE:1C:02 ]
   Decrypted correctly: 100%
```

line as a filter. As before, we locate the line which contains the parameters (username/password). Notice that on the packet bytes pane, the Frame tab and the Decrypted WEP data tab appear.

Table 1. *POST info request*

Key	Value
task:	login
username:	Admin
passwd:	l3tmeIn!

Protect from Snooping

All of the above examples show how easy it is to obtain sensitive data from snooping on a connection. The best way to prevent this is to encrypt the data that's being sent. The most known encryption methods are SSL (Secure Sockets Layer) and TLS (Transport Layer Security).

The Secure Socket Layer (SSL) and Transport Layer Security (TLS) are the most widely deployed security protocols used today. They are essentially protocols that provide a secure channel between two machines operating over the Internet or over an internal network. SSL Certificates have a key pair: a public and a private key. These keys work together to establish an encrypted connection. The certificate also contains what is called the "subject," which is the identity of the certificate/website owner.

FOTIS LIATSIS

Fotis Liatsis is a member and System/Network Administrator of the Greek Student Security Team – CampSec. Hi is also an OWASP member of the Greek Student Chapter and System Administrator of Hackademic. His interests include Ethical Hacking, Penetration Testing, Security/vulnerability research and applied Cryptography methods. He is a security enthusiast and Athcon fanatic. You can follow his Twitter (@liatsisfotis) or his blog for more information (http://www.liatsisfotis.com/).

Dynamic Memory Allocation in Unix Systems

It is not always possible, at compile time, to know how big to make all of our data structures. When we send an SQL query to the database, it may return twenty million rows, or it may return one.

The mechanism by which we persuade the operating system to give us memory on the fly, is called dynamically allocated memory. This memory is outside of the memory allocated to the process, in an area known as the 'heap', and our doorway into it, is a pointer to the first byte, returned by a function called `malloc()`.

When we see code containing calls to `malloc()`, it may be difficult to see what it all means, because of the way it has been written, so it may be advantageous to assemble this code, piece by piece.

The basic function, takes one argument, the number of bytes of memory required, and returns a pointer to the first byte of this, like this:

```
char *pointer;
int size = 1000000;
                         pointer = malloc(size);
```

Originally, `malloc()` used to return a pointer to char, since this pointed to one byte, as well as anything could, but this was too simple. These days, `malloc()` returns a pointer to 'void', which is exactly the same as a pointer to char, but the compiler won't let you use it, without a cast to your favorite data type.

Therefore, if we need a character array, in the midst of our computation, we would need to rewrite the call, to say:

```
                         pointer = (char *)malloc(size);
```

If `malloc()` fails, it returns a NULL pointer, which we are duty bound to check, so we code it as:

```
             if((pointer = (char *)malloc(size)) == NULL){
             printf("Memory allocation failed\n");
                        }
```

Now, it's starting to look ugly, and can be made downright hideous, by allocating an array of structures:

```
struct this{
    int one;
    int two;
    int three;
};
struct this *pointer;
int size = 1000000;
                       if((pointer = (struct this *)
   malloc(size * sizeof(struct this))) == NULL){
                             printf("Memory allocation
   failed\n");
                         }
```

Occasionally, it isn't possible to know in advance, exactly how much memory we need. We may be collecting data

from several different sources, to place in one array, and only know how much each source will provide, when we access it.

There is another function, which permits us to alter the amount of memory which we previously allocated with `malloc()`, called `realloc()`.

The `realloc()` function takes a pointer to a dynamically allocated block of memory, and a new size value, and returns a new pointer, to the extended memory:

```
char *pointer;
int newsize = 2000000;
      temp = (char *)realloc(pointer, size);
or, to be pedantic,
      if((temp = (char *)realloc(pointer, size)) == NULL){
      printf("Memory reallocation failed\n");
      }
```

If we need to use our original pointer, for cosmetic, or aesthetic reasons, to point to the new memory, we simply reassign it:

```
                    pointer = temp;
```

Very brave programmers, who have faith in the order in which operations are performed, can save the cost of a pointer, by recycling the original pointer:

```
                    if((pointer = (char *)realloc(pointer,
   size)) == NULL){
                         printf("Memory reallocation
   failed\n");
                    }
```

Don't do this because, down this road lies madness, and a few core dumps.

All of that was quite easy, really but, occasionally, we need an array of pointers to things which, themselves, are of variable size. For instance, we may be rifling the bank's database, looking for the loan payment records of all of its hapless customers. We don't know, in advance, how many customers there will be, or how many payments they made. We start with the declaration of the two dimensional pointer:

```
char **pointer;
```

Some programmers declare this kind of pointer as 'char *pointer[]', since this looks like a pointer to an array, but it may be more intuitive to think of this as a pointer to a pointer.

Our first task, is to make the pointer to a pointer point to more than one pointer, In other words, we need an array of pointers, of the correct length. At the moment, all we have, is eight bytes of memory, containing garbage. Those eight bytes need to contain the first address, of an array of addresses. We do this with `malloc()`:

Linked Lists

When we are collecting data, the obvious, and simplest way of doing so, is to declare a structure, then declare a pointer to its type, and malloc an instance. As we acquire more data, we simply realloc our array of structures, and tack the data on to the end.

For getting rows of data out of a database cursor, this is great, and you shouldn't consider any other approach. However, what happens if you want to remove the 154th data element from the array? Or, perhaps, insert the 154th element?

What if, you are storing data from several sources, like the roads on a map, which you need to attach to specific elements of your array, like the road junctions?

Not so simple.

Despite the mental picture conjured up by the word 'list', a linked list can be one dimensional, two dimensional or multi-dimensional. Apart from the street map mentioned above, another well-known application is an electronic circuit diagram, where there are components, connected by wires which, together, form a two-dimensional figure. Add to that, airline routes, railway systems, and the dynamically changing positions of pieces on a chessboard, and you get an idea of the usefulness of linked lists.

The Unix file system uses a linked list to map the blocks allocated to all of the files on a disk. As files are added, deleted, increase or decrease in size, the linked list is appropriately manipulated to reflect the current position.

Okay, so what, exactly, is a linked list?

One of my lecturers described linked lists as 'a hundred blind men, holding hands in the dark'.

To stretch the analogy a little further, we can add that two of the men have a little red light attached to their heads, so you can see them.

Basically, a linked list is a series of data structures, with a special data structure at the head, and another special data structure at the tail of the list.

Let's begin with a definition of the data structure.

```
struct queue {
    struct queue *fwd;
    struct queue *rev;
    char data[1024];
};
```

Ignoring the embedded data array, notice that there are two pointers, each to a type 'struct queue' within the data structure. One is a forward pointer (*fwd), and the other, a reverse pointer (*rev).

It is these pointers, which link the linked list. Since we are using a forward and a reverse pointer, this will be a doubly linked list, but for some applications, we can omit either pointer, and just create a singly linked list.

We'll only consider the doubly linked list, as the amount of extra effort to do so is minimal.

First, we need to define the special structures for the head and tail.

```
struct queue *head;
struct queue *tail;
```

Since these are currently pointers to nothing, let's initialize them to some real memory:

```
if((head = (struct queue *)malloc(sizeof(struct queue)))
    == NULL){
     printf("Can't allocate memory for head\n");
     return(-1);
}
if((tail = (struct queue *)malloc(sizeof(struct queue)))
    == NULL){
     printf("Can't allocate memory for tail\n");
     return(-1);
}
```

Now we have two blind men with lights on their heads, so we can see them, but they still can't see each other. Let's fix that. We take the fwd pointer of the head, and attach it to the tail, and the rev pointer of the tail, and attach it to the head.

```
head->fwd = tail;
tail->rev = head;
```

To identify the head and tail, we need to set the rev pointer of the head to NULL, and to do the same with the fwd pointer of the tail.

```
head->rev = NULL;
tail->fwd = NULL;
```

Now the two blind men have placed their free hand onto a wall, which gives a clue as to how we know we've reached either end, when we're searching the list.

Now, we have to add an element to our list, which we can do at the head or at the tail. This is usually done within a subroutine, imaginatively called `add_elmnt()` or something, since we don't want to repeat the code a few hundred times in our program.

First, we create an element

```
struct queue *elmnt;

if((elmnt = (struct queue *)malloc(sizeof(struct queue)))
    == NULL){
     printf("Can't allocate memory for elmnt\n");
     return(-1);
}
```

Then, to add this at the head, we do the following, in the following order. Changing the order may lead to attempts to attach to undefined pointers:

- We first take our rev pointer, and point it to the head, whose address we know.

  ```
  elmnt->rev = head;
  ```

- Then, we point our fwd pointer to the address pointed to by the head's fwd pointer.

  ```
  elmnt->fwd = head->fwd;
  ```

- Next, we take the rev pointer of the structure pointed to by the fwd pointer of the head, and point it to ourselves.

  ```
  elmnt-fwd->rev = elmnt;
  ```

- At this point, we are attached to both head and tail, and can safely detach the head's fwd pointer from the tail, and attach it to ourselves.

  ```
  head->fwd =elmnt;
  ```

Why did we do the acrobatics in the third step? Why not, instead, just say

```
tail->rev = elmnt;  ?
```

The answer is, that we only know the position of the tail before we add the first element. However, we always know that the rev pointer of the structure following the head points back to the head.

If we're adding our elements to the end of the list, we follow the same method, except that we only know the address of the tail:

```
elmnt->fwd = tail;
elmnt->rev = tail->rev;
elmnt->rev->fwd = elmnt;
tail->rev = elmnt;
```

Let us now assume that we have a list of a hundred elements, and we want to scan it.

We can't do an indexed scan, since we don't have an array, and we can't make any assumptions about the addresses of the elements, since malloc just grabs memory from wherever it's free.

We need a pointer to struct queue, to traverse the structures, so we define a cursor

```
struct queue *cursor;
```

Then, we set up a loop:

```
for(cursor = head-fwd; cursor->fwd-fwd != NULL; cursor =
    cursor->fwd){
        /* do loopy things */
}
```

The initialisation is obvious: we just need to start at the first element, past the head of the list. Occasionally, the head and tail contain extra elements, such as queue length etc, so it may be necessary to start with 'cursor = head', but we have no such need. The loop increment is equally obvious, in that the cursor sets its new address to that pointed to by the current element.

The loop termination conditions may not be so obvious. Why not just say 'cursor != tail'? Well, you can. However, it is not a good habit to get into, since some loops may have conditions within them, which cause the cursor to increment by more than one element. Down that road lies 'segmentation error – core dumped'...

Looking for a NULL fwd pointer is a guarantee that you've reached the end of the list, since only the tail has it set to NULL.

How about searching in reverse? Easy.

```
for(cursor = tail->rev; cursor->rev->rev != NULL; cursor =
    cursor->rev){
        /* do loopy things */
}
```

Now that we can insert elements, and create a long list, then search our list, this just leaves us with the task of deleting an element.

We need to take the same amount of care with deleting, as we took with adding an element. For the sake of example, let's say we want to delete any element with an empty data element in the queue structure;

```
for(cursor = head-fwd; cursor->fwd-fwd != NULL; cursor =
    cursor->fwd){
        if(cursor->data[0] == 0x00){
            cursor->fwd->rev = cusor->rev;
            cursor->rev->fwd = cursor->fwd;
            free(cursor);
        }
}
```

We take the rev pointer of the structure pointed to by our fwd pointer, and point it at the address being pointed to by our rev pointer. Next, we take the fwd pointer of the structure being pointed to by our rev pointer, and point it at the address being pointed to by our fwd pointer.

This has now bypassed our current element, so we can free it. Right? Well, the cursor address is still the same as that of the original element so, yes, we can.

However, what happens when we get back to the top of the loop? It'll try and set cursor to cursor->fwd. This will work – most of the time.

The problem is, that we just freed that piece of memory, which gives the operating system permission to give it to someone else. On an idle system (like the development machine), nothing will happen, and the loop will run to completion but, on a busy system (like production) another process might snatch that piece of memory, leaving our cursor to jump into the weeds, somewhere on the heap, and the testers will call you out in the middle of the night to fix it.

You could decide that you can live with the memory leak, and omit the `free()` call, in which case, you should firmly close this page, and seek an alternative career.

To do it properly, what you need, is a second cursor.

```
struct queue *sentry;

for(cursor = head-fwd, sentry = cursor; cursor->fwd-fwd !=
    NULL; cursor = cursor->fwd){
        if(cursor->data[0] == 0x00){
            sentry = cursor->rev;
            cursor->fwd->rev = cusor->rev;
            cursor->rev->fwd = cursor->fwd;
            free(cursor);
            cursor = sentry;
        }
}
```

Now, let's see what happens.

As soon as we've found the element we wish to delete, we set sentry to the previous element. When we've deleted our element from the list, and freed its memory, we set cursor to the same address as sentry, which is the element before the current one. The loop now advances the cursor, correctly, to the next element.

As we mentioned earlier, linked lists can be multi-dimensional. To create a two-dimensional list, suitable for creating matrices, maps, and other topological representations, we only need to change the basic element.

```
struct elmnt (
    struct elmnt *fwd;
    struct elmnt *rev;
    struct elmnt *up;
    struct elmnt *dn;
    char data[1024];
}
```

Now, instead of just a forward and a reverse pointer, we have an up and a down pointer, as well.

The process of adding an element now also includes setting the two latter. If the element being added is just another linear element, we set the up and dn pointers to NULL but, if it is a branch point, we have to set them to point up to the newly added structure, and back down to the branch point.

Let's say we already have our linear linked list, and we wish to add one element above, and another below the first element after the head.

```
elmnt->dn = head->fwd;
head->fwd->up = elmnt;
head->fwd->dn = NULL;
elmnt->up = NULL;
```

Note that we leave no trailing pointers, but terminate them with a NULL, so we can find the end of the branch.

Next, we add a new element below the first element after the head.

```
elmnt->up = head->fwd;
head->fwd->dn = elmnt;
head->fwd->dn->dn = NULL;
```

Note that we don't set the head->fwd->up pointer to NULL, as we just added an element there.

Traversing such a list will require two cursors, in two nested loops. The main loop traverses the list in a horizontal direction, with hcursor, while the two inner loops traverse the branches vertically up or down, with vcursor.

```
for(hcursor = head-fwd; hcursor->fwd-fwd != NULL; hcursor
   = hcursor->fwd){
    if(hcursor->up != NULL){
        for(vcursor = hcursor; vcursor->up != NULL;
   vcursor = vcursor->up){
              /* traverse the upward bound list */
        }
        if(hcursor->dn != NULL){
            for(vcursor = hcursor; vcursor->dn != NULL;
   vcursor = vcursor->dn){
              /* traverse the downward bound list */
        }
    }
}
```

Three dimensional linked lists work in exactly the same way, with an element defined as

```
struct elmnt (
    struct elmnt *fwd;
    struct elmnt *rev;
    struct elmnt *up;
    struct elmnt *dn;
    struct elmnt *out;
    struct elmnt *in;
    char data[1024];
}
```

where 'out' and 'in' are the z-axis pointers.

It is left as an exercise for the reader, to design a function to add such an element to a linked list, and then to define a traversal function.

MARK SITKOWSKI

Mark Sitkowski C.Eng, M.I.E.E Consultant to Forticom Security

Technology makes a wonderful slave but a cruel master. Both Amazon and Tesco, major retailers in the UK and worldwide have been severely criticised in the media for the use of technology to control and monitor staff excessively. As IT professionals, where do we draw the ethical line in the sand?

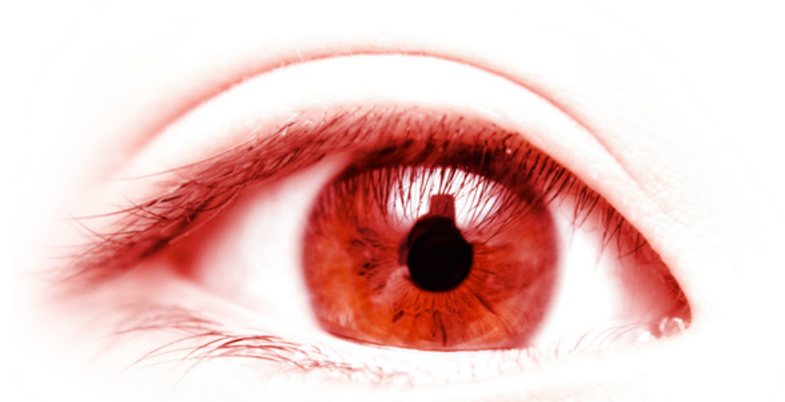
To quote Albert Einstein, "Technological progress is like an axe in the hands of a pathological criminal." Time and again throughout history, as a society we have seen the positive contributions made by innovators, creatives, engineers, architects and humanitarians perverted and used for immoral if not evil ends. Tempting though it would be to take Einstein's quote and neatly assign to the technologists the role of the angels and to the politicians, bankers, society or whoever else the role of the pathological criminal, this would be far too simplistic. As far as I am concerned, the actions of black-hat hackers, spammers and the various other forms of Internet low-life are definitely criminal if not pathological. Of course, we must make allowances for the uneducated and the unaware, and I do not include here the average end user who has a compromised PC due to poor web hygiene. No, we are talking about those whose hearts are dark and who choose to use technology for their own agenda, rather than for the benefit of all.

Traditionally, the guru was party to esoteric knowledge shared with others either for financial, spiritual or social status. The first rule for the guru was the protection of knowledge and wisdom, as it was widely understood that the value of the guru would be inversely proportional to the number of people who were cognisant to the "magic". Essentially, the same morality exists today in the form of the established professions – Doctors, Lawyers, Architects etc. – the amount of studying, self-sacrifice and knowledge that is required to achieve qualification and recognition is great, so the profession then erects barriers to those that are not initiated. This in turn leads to separation within society, between those with the knowledge and as a consequence – power – and those that do not. This has led to cries from the "have nots" of injustice, and so the political ideologies of Marxism, Communism, Maoism, Stalinism, Socialism etc. gained traction and political credence in the 20th century. Irrespective of the basis of these riches, whether they be intellectual, financial, or physical, there were secrets to keep, professional relationships to be nurtured and at all costs the status quo to be maintained.

Aside from political argument as to whether or not Capitalism or any other doctrine is superior, the second rule for the guru is do not whistle-blow. Ever. The consequences of being an initiate and sharing "dirty washing in public" range from censure, character assassination to potentially death depending on the quality, importance and potential embarrassment caused by the information being shared. Just ask Frank Serpico. Unfortunately we cannot ask Karen Silkwood. Of course, if "leaking" information is useful to discrediting another guru, often this will be encouraged.

So I have no problem at all of awarding Edward Snowden the author's "IT Man of the year" award for courage, honesty and integrity but qualified with a very small pinch of salt. While it is difficult to get to the bottom of any spook-based operation, especially taking into account the incestuous relationship the media (including the alternative media) have with the security services, it is hard to

reconcile on a pragmatic basis why ES chose to seek asylum in Russia. Maybe it was the harsh hand of fate, the bitter cup of circumstance that placed him in these circumstances. Unless this becomes public knowledge, or we manage to share a cup or two of coffee I doubt I will ever know. But if I was in his shoes, I would have chosen a host that couldn't potentially change his role from truth-teller to political pawn a la the exchanges that happened on the borders of East and West Germany during the cold war. We mustn't judge though – as far as I am concerned to discuss, please feel free to email me at *me@merville.co.uk.*). Others are more comfortable bearing their heart in short bursts. I aim for 1000 words. Maybe, I am a dinosaur, but as I mentioned earlier context is everything, and that is why every guru has to take his personal path to enlightenment. Only you know from your personal value system if the project you are working on is a threat. Does it pass the smell factor? How uneasy do you feel? Could you justify it in front of your manager? The CEO? The shareholders? Society? The universe? God?



ES has made a tremendous sacrifice and we must honour that irrespective of the geopolitical rhetoric. In my book, truth-teller, whether communist, fascist or capitalist must be applauded wholeheartedly.

But lets get back to reality, rather than a media frenzy of accusation and counter accusation. The problem with committed IT professionals (and I use the word committed here in the sense that we are passionate rather than candidates for the lunatic asylum) is that what we are involved with is often in the scale of rocket science, nuclear physics or whatever. A few thousand lines of code can change lives. Our product can be the stiletto that is used to shave 20% off the staffing levels of an organisation, or maybe as system administrators we can be asked to forget major "ethical hiccups". And some of us write code for nuclear weapons guidance systems. When you are submerged in lines of code, caught in the political management crossfire with a serious deadline due, or just burnt out with the whole shebang, it is important to remember the context, despite how difficult that is to do.

Like all of society, IT has its mix of extroverts and introverts. Personally, I prefer quality over quantity, so I spend my time writing long leader columns that will hopefully entertain and communicate rather than lots of spurious noise on Facebook and Twitter. Sheesh, I don't even have a blog. So in Internet terms, I am probably a confirmed introvert (I do occasionally reply to emails. If you have any constructive comments on these columns, or would like

To be honest, I feel sorry for the coders and techs involved in the Amazon and Tesco projects. Payback in the form of negative media exposure, no matter how distanced you are from the source or target is never pleasant. At the time, everything was probably justified from a management and project perspective, but naturally hindsight has 20-20 vision. In all my years as a tech, apart from those leaning towards or in management, I have never met an IT specialist who wanted to see jobs lost or benefits reduced by the application of technology. Maybe I have worked with too many idealists, but we all wanted to make things better. Safer. More productive. Less stressful. More fun. And at the same time make an honest buck. So let's raise our glasses in New Year 2014 to the Snowdens, Assanges, Tesco and Amazon employees who have had the courage to blow the whistle. And may they be our encouragement to do likewise as we enter deeper into the age of the pathological criminal.

ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid-eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.